Intelligent Decision Making for Camera Views and Behaviours

Development of an Intelligent Camera System for Games Capable of Both Including Important Game Elements and Respecting the Player's Intentional Views

Martin Philip Hansen-Schwartz, 170782-2569
Supervisor: Arnav Harish Jhala

Media Technology and Games
IT-University of Copenhagen

A thesis submitted for the degree of

MSc in IT: Media Technology and Games

August 2009

Abstract

Many computer games, especially games of the adventure genre, require the player to solve various tasks or puzzles to progress in the game. Depending on the player's level of experience in playing games, the player might need help or hints on how to solve a certain task/puzzle. One way of giving hints to the player is using the camera as a helping tool. In the history of movies, cinematographic principles have been used for many years to emphasize and dramatize specific elements on the screen. These principles can be used in the same way to help the player notice current important elements in the game. However, not all players play the same way, for example sometimes the player wants to explore rather than complete the game, which of course the player should be allowed to do. A camera system, in dynamic environments should adapt to provide cinematic views to players while giving them freedom to control their point of view.

A method for automated camera placement is presented, which is based on previous work by Cozic [1] for managing the view between the camera system's optimal view (the camera hints) and the player's intentional view by using fuzzy logic to choose view. An implementation of the adaptive camera is presented in a complete game with preliminary comparison of current techniques between current state of the art.

Contents

List of Figures **viii**

List of Tables **xi**

1 Introduction **1**

1.1 Problem 1

1.1.1 Aim 1

1.1.2 Goals 2

1.2 Implemented Approach 2

1.3 Practical Information 3

2 Related Work **4**

2.1 Techniques for Camera Control in Virtual Environments 4

2.1.1 Industry Techniques 5

2.1.1.1 Benefits 5

2.1.1.2 Drawbacks 6

2.1.2 Academic Techniques 6

2.1.2.1 CamDroid 6

2.1.2.2 ConstraintCam 7

2.1.2.3 The Virtual Cinematographer 8

2.1.2.4 Visual Composition as Optimisation 8

2.1.3 Approaches Targeted at Computer Games 9

2.1.3.1 A Camera Engine for Computer Games 9

2.1.3.2 Constraint-based Autonomous 3D Camera System . . . 11

2.1.3.3 Autonomous Real-time Camera Agents 12

2.1.3.4 Automated Cinematography for Games 12

2.1.4 Summary 13

2.2 Camera Systems in Games 13

2.2.1 Fixed Camera Systems 13

2.2.2 Tracking Camera Systems 14

2.2.3 Interactive Camera Systems 15

2.3 Choice of Camera System: Discussion 15

3 Overview of Related AI Techniques 17
3.1 Finite State Machines 17
3.2 Genetic Algorithms 18
3.3 Neural Networks 19
3.4 Fuzzy Logic 20
3.5 Choice of AI Technique: Discussion 21

4 Fuzzy Logic Decision Maker 23
4.0.1 Fuzzy Sets 23
4.0.1.1 Membership functions 24
4.0.1.2 Fuzzy Set Operators 24
4.0.2 Fuzzy Linguistic Variables 26
4.0.3 Fuzzy Rules 26
4.0.3.1 Fuzzy Inference 26
4.0.3.2 Defuzzification 27
4.0.4 The Combs Method 28
4.1 Fuzzy Logic Setup in FLCam 29
4.1.1 FLVs 29
4.1.2 Fuzzy Rules 29

5 The Implementation on the Game Engine 36
5.1 Use an Existing Game or Create a New One? 36
5.2 General System Structure 37
5.3 The Algorithm 38
5.4 Scene Object Events 39
5.5 Shot Contribution Update 40
5.6 Current Shot Update 41
5.7 Camera Parameters Setup 41
5.7.1 Camera Shot Setup 41
5.7.2 Setup of the line of action 42
5.7.3 Choosing Camera Behaviour Using Fuzzy Logic 43
5.7.4 Choosing Shot Weight Using Fuzzy Logic 44
5.7.5 Constraint Solver 45
5.7.6 Calculate Frame Coherence Using Fuzzy Logic 46
5.7.7 Interpolation Between the Old and New Camera Setup 47
5.8 Scene Object Interface 47

6 Example Scenarios 48
6.1 Descriptions of the Camera Systems Used 48
6.2 Comparisons 50
6.2.1 Example Scenario One 50
6.2.2 Example Scenario Two 51
6.2.3 Example Scenario Three 52

6.2.4 Example Scenario Four . . . 52
6.2.5 Example Scenario Five . . . 53
6.3 Summary . . . 54

7 Discussions **55**
7.1 Critical Assessment . . . 55
7.2 Future Work . . . 56
7.3 Summary . . . 57

A Fuzzy System Setup for FLCam **58**
A.1 Fuzzy Rules Used in FLCam . . . 58
A.2 FLVs Used in FLCam . . . 61

B Object & Shot Rules **63**

C Puzzle Bloom Screenshots **67**

D Rules for the Crisp Decision Maker **74**

E Summary of the Project Process **76**

Glossary **78**

References **79**

List of Figures

2.1 Screenshots from Metal Gear Solid 2 using fixed camera 14
2.2 Bad camera view from games using fixed camera 14
2.3 Screenshots from Tomb Raider 15

3.1 A feedforward neural network 19
3.2 Fuzzy rule-based inference 21

4.1 A graphical representation of a crisp set representing categories for angle values in degrees between 0-180° 24
4.2 Examples of member functions for fuzzy sets 25
4.3 Dumb, Average, and Clever as fuzzy sets. The dotted line represents an angle of 115 degrees, and its points of intersection with the sets *Large* and *Medium* represent its degree of membership in those sets. 25
4.4 (a) shows the degree of membership of the value 30 to the set $Angle_VerySmall$ and (b) shows the degree of membership of the value 9 in the set $Dist_Close$ 30
4.5 (a) shows the degree of membership of the value 30 to the set $Angle_VerySmall$ and (b) shows the degree of membership of the value 9 in the set $Dist_Medium$ 30
4.6 (a) shows the degree of membership of the value 30 to the set $Angle_VerySmall$ and (b) shows the degree of membership of the value 9 in the set $Dist_Far$ 31
4.7 The FAM for the shot weight rule base given the input values target angle = 30, target distance = 9 and the average shot contribution = 0.4. The shaded cells highlight rules that have been fired. 32
4.8 The inferred results of processing the rule set for shot weight 33
4.9 Combining the conclusions 34
4.10 The graphical outputs of shot weight 0.2 (a), 0.5 (b), 0.85 (c) and 0.57 (d) 35

5.1 Screenshot from Puzzle Bloom, the game which will be used as the test bed 37
5.2 General system structure and flow of information 38
5.3 Scene Object Events 40
5.4 Top-down view example of how the camera system sets up the camera shots including scene objects 42

5.5 Example of how the line of action is calculated when SOs are present in the current shot 43

5.6 An example situation of how the fuzzy decision maker uses the look-at points (p1 and p2), the direction vectors from the lines of action (d1 and d2) and the radii (r1 and r2) from the two camera setups with and without the scene objects to find the new camera setup (p, d and r) . . 45

6.1 Example scenario 1 50

6.2 Example scenario 2 51

6.3 Example scenario 3 52

6.4 Example scenario 4 53

6.5 Example scenario 5 54

A.1 (a) is the angle between the direction vector from the avatar's position to the look-at point and the avatar's forward vector and (b) is the distance between the look-at point the avatar's predicted future position 61

A.2 (a) is the average shot contribution of all the SOs in the current shot and (b) is the interpolation value between the optimal view and the player's intentional view, where 0 is completely the optimal view and 1 is completely the player's view 61

A.3 (a) is the angle between the camera's current forward vector and the camera's ideal forward vector, (b) is the distance between the camera's current position and the camera's ideal position, (c) is the frame coherence weight for the camera, (d) is the distance weight for the camera, (e) is the height weight for the camera and (f) is the orientation weight for the camera 62

C.1 Extra scenario 1 ((a) - (d)), the first puzzle of the game where the player learns how to switch to another creature. Extra scenario 2 ((e) - (h)), demonstrates how well the camera systems adapt to when important objects are almost behind the player. 68

C.2 Extra scenario 3 ((a) - (d)) has both the door and the other creature as important objects because the player needs to switch to the other creature to get through the door. Extra scenario 4 ((e) - (h)), shows the newly opened door to the next part of the game as an important object. 69

C.3 Extra scenario 5 ((a) - (d)) demonstrates how well the camera systems adapt to when important objects are almost perpendicular to the player's viewing direction. Extra scenario 6 ((e) - (h)) demonstrates how it is only the camera system with the fuzzy decision maker, which gives a perfect view of the important object in the scene when there is a large angle and distance to the important object. 70

C.4 Extra scenario 7 ((a) - (d)) shows the player's first experience with the crate (which is the important object) and the player is forced to learn how the crate can be used. Extra scenario 8 ((e) - (h)) shows how the fuzzy camera system gives the best view of the important creature which the player needs to switch to next. 71
C.5 Extra scenario 9 ((a) - (d)) shows how the fuzzy camera system gives the best view of the creature which becomes important once the button is pressed. Extra scenario 10 ((e) - (h)) shows how the fuzzy camera system gives the best view of the important creature which the player needs to switch to. 72
C.6 Extra scenario 11 ((a) - (d)) demonstrates how well the camera systems show the door which becomes important when the button is pressed. . . 73

List of Tables

4.1 Rules Required (traditional) vs Rules Required (Combs) 28
4.2 Summary of example 32
4.3 Representative values of the sets comprising the output manifold 35

5.1 Scene Object Interface 47

Chapter 1

Introduction

Computer games – in particular adventure games – contain complex tasks and puzzles that are often hard to solve for average players. Traditional hint mechanisms are text-based and break the player's immersion into the game. Hinting the player can be done for example by placing important elements on their own to make them more obvious or dominating, which sometimes can look a bit odd in the game world. One way to avoid this and still help the player is to use the camera as a tool using cinematographic principles just as in films to give hints to the player. In this way, important elements can be placed in crowded areas and still make the player take notice of the specific element, because the cinematographic principles will emphasize the important element to the player. Such techniques for visual emphasis must not interfere with the player's agency.

1.1 Problem

This thesis will explore the possibilities for developing a decision making method in deciding on a view to use by taking the camera system's optimal view (the camera hints) and the player's own intentional view as inputs. In this way it is hoped that the camera hints will not break the experience of the player's intentional goals but still allow the hints to be useful.

1.1.1 Aim

The aim of this project is to develop a method for a camera system, which takes two camera views as input more precisely one camera view, which is most optimal from the camera system's point of view and another camera view, which is the player's own

intentional view. This method should be able to intelligently decide on a camera view which is compromised between the two input views. To fulfill this aim, the following goals must be achieved:

1.1.2 Goals

- Compare previous works on real-time camera control from the camera control research area and find the one most suitable to extend to create a decision making method as described above
- Compare different AI techniques and find the one most suitable to use for developing a decision maker
- Design and implement the decision maker method
- Integrate the decision maker method with a suitable game
- Evaluate the decision maker method by comparing the different camera views from when playing the game with and without the method

1.2 Implemented Approach

To begin with, brief descriptions are given of the related works done in the camera control research area. This is followed by an analysis of the problem domain describing different types of camera systems used in games and some general AI techniques. From this a type of camera system is chosen on which to base the camera system on and an AI technique is chosen to use for the decision maker method.

The next part of the thesis introduces fuzzy logic and reasoning of why this AI technique is chosen. The setup of the fuzzy system is described in detail as well as an example of how fuzzy logic works. Furthermore, it is discussed whether to integrate the system and method with an already existing game or whether a new game should be made from scratch. This is followed by the details of how the camera system is implemented on the game engine (Unity Technologies game engine). This includes details of the algorithm used by the camera system and the techniques used to calculate the camera views.

In the last part of the thesis the decision making method will be evaluated by comparing the camera views seen in similar situations of the game when playing with and without the method.

1.3 Practical Information

Please refer to the glossary for a definition of each cinematographic term used in this thesis.

A digital copy of this report can be found at:
www.itu.dk/people/mschwartz/portfolio/mschwartz_thesis.pdf

and a copy of the source for the camera system can be found at:
www.itu.dk/people/mschwartz/cameratest/flcam_src.zip

The modified version of Puzzle Bloom (including FLCam) can be played at:
www.itu.dk/people/mschwartz/cameratest/PuzzleBloom.html

Chapter 2

Related Work

This chapter presents some of the related work in the research area of camera control and a description of state of the art systems in the industry.

2.1 Techniques for Camera Control in Virtual Environments

In this section a description of some of the techniques used by the game industry and a brief description of the techniques, theories and related systems in the research field of camera control in virtual environments (not necessarily game environments) is given. Christie and Olivier [2] classifies the solving mechanisms of camera control into four classes:

- **Algebraic systems** represent the problem in vector algebra and directly compute a solution;
- **Interactive systems** set up cameras in response to user interaction through an input device and provide immediate feedback to the user;
- **Reactive real-time systems** utilize motion-planning algorithms from robotics to reactively move cameras in dynamic environments;
- **Optimization and constraint-based systems** represent camera properties as con- strains and optimize the ob jective function through a variety of solving processes that widely differ in their properties (e.g. incompleteness, softness, etc.)

This section focuses most on the last two classes with focus on systems that can be applied to real-time interactive environments, but a brief look at the techniques the industry is currently using is also given. Due to the the nature of the industry, documentation on techniques for published games is difficult to obtain. The section below describes known and documented approaches used in commercially available games.

2.1.1 Industry Techniques

In game environments, the industry tends to use very computationally efficient methods as computer games need to be optimised to give the highest frame-rate possible. A simple and common camera control technique usually uses polar [3] or spherical [4] coordinates to position the camera in direct relation to the camera. In both approaches (polar and spherical) the camera will be maintained at a fixed distance, height and orientation from the target object (the player avatar). These properties can be considered *hard* constraints, because the camera will maintain these viewing properties every frame. This creates usability problems (over-rotation especially in interactive control), therefore spring (or damping) systems have to be used to keep the camera's motion more convincing and to allow the target to be controllable.

Another method of camera control [5] which is a bit more complex makes use of steering behaviours [6]. Steering behaviours provide a mathematical method for orienting an object in relation to another object. This method is perfect for camera systems, whose fundamental goal is to track a target through an environment. The main issues with this method is that it suffers from the need for spring systems to generate convincing motion and avoid 'over-shooting' the target.

A final method (used particular in the first two Tomb Raider games) which this paper will mention on industry techniques, involves quaternion interpolation for camera movements [7]. This works by using Spherical Linear intERPolation (SLERP) to smoothly rotate the camera. When the target object (changes orientation), the camera is provided with the new orientation and performs a SLERP to smoothly rotate towards the new orientation. This is essentially a sophisticated spring system to handle the over-rotation problem that occurs using an regular polar or spherical coordinate camera system.

2.1.1.1 Benefits

The methods are computationally efficient because they use very simple relationships with the target and do not use any *real* artificial intelligence techniques (i.e. there is no usage of any complex search algorithms).

2.1.1.2 Drawbacks

Because of the very simple relationships with the target these methods become very inflexible for changing domains (for example game genres) and purposes. Because of more complex game environments in recent games with the possibility to access very tight areas there is need for more and more advanced spring/damping systems to achieve convincing movement, and these are prone to oscillation. In general it is very complex to configure the spring/damping systems and typically requires extraordinary number of parameters ($\approx$ 100 [8]).Additionally because of their rigid structure it is close to impossible to extend these systems capabilities.

The methods used in academia are more sophisticated and some of the most approaches in the camera control research will be described below.

2.1.2 Academic Techniques

2.1.2.1 CamDroid

CamDroid [9] is an early camera control system developed by Drucker and Zeltzer. The system is based on improvements of their CINEMA system [10]. It introduces the notion of 'camera modules'. Each module represents an encapsulation of primitives constraints, which prevent the user from dealing with low level interactions to control the camera.

The authors compare the concept of a camera module with the concept of a shot in cinematography. A shot is a part of scene which is continuously filmed without stopping and thereby a shot represents a continuity of all the camera parameters over the shot's time length. In the same way a camera module requires the same continuity as a shot does, but it also requires a continuity of control of the camera. This requirement is used to blur the distinction of transition between two different modules.

The components of a generic camera module consists of an initialiser, a controller, a constraint list and a local state. The local state contains the camera position, view, "up" vector and field of view. The initialiser uses the previous state to set up this module's camera parameters. The controller translates the user inputs either directly into the camera state of into constraints. The constraint list contains all the constraints which must be satisfied during the module's period of time. All the constraints are combined with a constraint solver and solved by a camera optimiser based on the CFSQP (Feasible Sequential Quadratic Programming coded in C) package.

CamDroid was an acceptable initial attempt controlling virtual cameras using constraint satisfaction, but it had some major drawbacks. It does not offer a automised solution on how to handle over-constrainted situations which can occur frequently in dynamic environments because of autonomy of dynamic objects. This issues were ad-

dressed by Bares' camera system ContraintCam [11].

2.1.2.2 ConstraintCam

Bares' ConstraintCam [11] is a real-time camera planner for complex 3D virtual environments. It creates camera shots which satisfy user-specified goals in the form of geometric composition constraints. To solve CamDroid's issue of constraint failures ConstraintCam employs partial constraint satisfaction. If the system fails to satisfy all the constraints of the original problem, then the system relaxes weak constraints and if it is necessary a single shot will be decomposed to create a set of camera placements that can be composed in multiple viewports [12] to provide an alternative solution to the user (which is more likely a level designer to script the camera control and not a player). ConstraintCam is based on the following design guidelines:

- *Arbitary Viewing Goals:* Viewers can update the constraint set at any time or request alternate views of any subset of initially specied constraints.
- *Environmental Complexity:* It supports planning for solutions in complex environments by taking into consideration high geometric complexity, holes, or transparent surfaces.
- *World Non-Interference:* Limited support for world manipulation to articially create good compositions. For example, the system can opt to move an ob ject that is out of position in the shot composition.
- *Failure Handling:* Systematic methods such as the ConstraintCam solvers produce sub-optimal shots by relaxing constraints after taking into consideration their relative priority.

The interface in ConstraintCam allows the user to select which subjects(s) to be viewed at any given time, specify the vantage constraints for each subject, select the cinematic style or pace and select the number of inset viewport used to present multi-shot solutions. If the user chooses constraints which can not all be satisfied at the same time then the system will use partial constraint satisfaction. Each constraint is weighted so that during relaxation constraints with lower strength will be disabled before those of greater strength.

If no solutions can be found for the given constraints, then the constraint solver tries to find an alternate solution. The constraint solver will first try to identify which combination of constraints that are incompatible, this is accomplished by constructing a *incompatible constraints pair graph*. Next it attempts to find a maximal solution satisfying as many of the higher strength constraints as possible by relaxing weaker constraints until no incompatible constraint pairs remain.

In [13] Bares et al. introduced constraint weighting on the visual constraints to provide a mechanism for preference of certain visual properties. The system uses a modified CSP representation with camera position (x,y,z), view (xv,yv,zy), FOV (field of view) and up vector (dx,dy,dz) defined as the variables. The measure of satisfaction of a constraint is calculated as a fractional value between 0 and 1. The total satisfaction measure for a given solution is computed using the weighted sum of the satisfaction values of the all the constraints.

$$satisfaction = \sum_{i=1}^{n}(P_i \times S_i) \tag{2.1}$$

The constraint solver for this approach is a generate-and-test method. The solver constructs regions of space using a 20x20x20 grid to test if a point in the grid fails to satisfy the camera positioning constraints. The solver mechanism of this approach was refined in [14] by refining the the scale of the grid recursively. The top five points of each pass were retained and used for the next pass. The process is performed until a suitable solution is found or the the search has exhausted its time or moves. The deeper the search gets the more evaluations for potential solution need to be made, which is why this approach often requires a lot of computational time and thereby makes this approach unsuitable real-time environments.

2.1.2.3 The Virtual Cinematographer

He *et al.* [15] use "idioms" to describe the behaviour of the camera for each given type of scene. The rules of cinematography "are codified as a hierarchical finite state machine" which "controls camera placements and shot transitions automatically". The system also uses a library of camera modules, which contains information such as the position of the characters on screen and the shot scale. In its current state, the system can successfully capture specific scenes such as a conversation between two or three actors, but the authors mention that the system can fail due to unexpected occlusions. The other noteworthy limitation is that it can only deal with forecast situations this making it difficult to adapt it to complex virtual environments, which is why [15] would not work great in unpredictable worlds like computer games.

2.1.2.4 Visual Composition as Optimisation

In [16] Halper *et al.* use a genetic algorithm to automatically generate the optimal camera placement within a virtual environment given *shot properties* as defined by the user. These include required position, size, orientation and visibility of scene elements relative to the *viewplane* (the plane that is perpendicular to the viewing direction),

viewport (the section of the viewplane that is viewed) and the *viewpoint* (the location of the camera).

With the shot properties defined, the optimal shot is found by a genetic algorithm. Genetic algorithms model the solving of a given problem as a "competition among a population of evolving candidate problem solutions. A 'fitness' function evaluates each solution to decide whether it will contribute to the next generation of solutions. Then, through operations analogous to gene transfer in sexual reproduction, the algorithm creates a new population of candidate solutions" [17]. Halper *et al.* encode the camera's position (x,y and z), the camera's orientation (roll, pitch and yaw) and its field-of-view, in the chromosomes of each gene and randomly distribute a population of these in a bounded search space. The overall fitness values is derived from a combination of the fitness values of all the genes based on their adherence to the shot properties defined by the user. To determine the fitness values the system evaluates the following image properties:

- The location of the centre of a scene element
- The projected extents of scene elements
- The visibility of scene elements

After an evaluation the fittest 90% of the population of genes survive into the next generation. The other 10% are regenerated by random crossover and/or mutation of the genes. Generations continue to be produced until the user aborts or an optimal combination of chromosomes is found for the camera state, i.e. the evaluation of the camera state has determined it to best conform to the user's requirements. Although this is a sophisticated method for producing an appropriate shot given some visual requirements, it is not designed to operate in a real-time application and so it is not suited to this thesis' requirements.

2.1.3 Approaches Targeted at Computer Games

All the approaches described in the previous section were not developed with the direct intention of using it in a real-time dynamic environment like computer games.

2.1.3.1 A Camera Engine for Computer Games

Halper et al. are the first to target a camera control approach in academia in the domain of computer games. The approach [18] implements a camera engine based on targeted objects both in 2D and 3D spaces. They have based the engine of three basic requirements:

- **flexibility:** must use parameterised techniques and versitility in dening constraint specications in order to adapt to the output of events from the action module.
- **information:** the more the camera knows about what is going on in the world, the better. There is need of event calls, information from actors, player motivations, and visual goals.
- **satisfaction:** a best-t solution will not always be present. Therefore, there is need of partial satisfaction solutions, and incorporate adaptive degradation, so that it is possible to at least convey the most important visual cues at any given time.

The authors propose a full set of properties on the objects of the game, varying from height angles, angles of interest, size, visibility and positioning on the screen. Since events generated in the game by the story and actions of the game may change at any given time, all constraints must be reevaluated for each new situation. In contrast to a purely reactive application of constraints like Bares [12] where camera always tries to jump to the global best-fit solution, Halper et al. introduce the notion of frame-coherence, in other words try avoid "jumpiness" in the camera motion.

The way frame-coherence is maintained is by using an algebraic incremental constraint solver to compute the new camera configurations given the current camera state. The system also uses relaxation techniques. If all constraints cannot be satisfied only certain constraints will be solved and the camera state will then be modified to accommodate the remaining constraints. Lookahead methods are used by the system to adjust the camera parameters for future situations by estimating future object states based on their past trajectories and velocity information.

To address occlusion the system takes advantage of graphics hardware. It renders the scene in hardware stencil buffers with a colour associated to each object to evaluate the number and extent of occluding objects. The major drawback of using graphics hardware is the dependency of the underlying hardware and thereby you lose generality and portability, but besides from the raw performance of hardware rendering there are also some other advantages:

- the use of very low resolution buffers (32x32 pixels) is sufcient to deal with occlusions
- such techniques are independent of the internal representation of the objects (no need of boundaries for the objects of the scene)
- rendering the scene without any use of bounding volumes simplication ensures that exact occlusions are computed (modulo the resolution of the buffers)

Though this approach has addressed the challenges of maintaining smooth camera movement and acceleration and occlusion avoidance and succeeded, Bourne claims in [19] that there is still no generic solution to problem of controlling an autonomous camera.

2.1.3.2 Constraint-based Autonomous 3D Camera System

In [20] Bourne et al. designed a novel autonomous camera system which addresses the unique and difficult challenges of interactive digital entertainment (IDE) here under computer games. The authors' research are directed towards achieving an effective unified methodology to handle camera control and visibility maintenance. Their research is motivated by the deficiencies in the camera control methods used within the IDE industry and the authors propose some guidelines for an autonomous camera system for IDE needs to have:

- **Real-time.** The camera must be responsive to user input and operate under interactive time constraints.
- **Competent.** The camera must use a exible architecture to adapt to changing demands required by games.
- **Extendable.** The camera must be responsive to user input and operate under interactive time constraints.
- **Dynamic.** The camera must deal with complex and dynamic environments.
- **Environment independence.** The camera must be able to deal with a variety of 3D worlds commonly found in games.
- **Ease of implementation.** The camera must not require pre-processing of the environment.
- **Cinematic.** The camera must support cinematic functionality.

The system uses a constraint solver which exploit the spatial structure of the problem and thereby enabling it to be used in real-time environments. It uses a weighted constraint satisfaction problem (CSP) framework with reactive which means that each frame of animation is treated as a new CSP to be solved just like in [18]. This allows the camera to be used in highly dynamic and interactive environments without additional dependence on the efficiency and robustness of predictive systems. The following three constraints are used to control the camera:

- **Height.** Maintains a relative height relationship with the target.

- **Distance.** Maintains a relative distance relationship with the target.
- **Orientation.** Maintains a relative angular alignment between the camera view vector and the target facing vector.

Just as in [18] Bourne uses frame-coherence to maintain a smooth camera movement, but instead of using an algebraic incremental constraint solver Bourne introduces a *frame-coherence* constraint. This constraint evaluates the difference between the distance the camera wants to move in the current frame and distance moved in the previous frame. A sliding octree has been applied to solve the constraints. By using this method the problem structure can be exploited and has been proven to perform better than both local and complete search.

2.1.3.3 Autonomous Real-time Camera Agents

Hornung *et al.* [21] have designed a camera system that improves Half-Life cut-scenes by choosing cinematographically appropriate shots. Narrative events are sent to the camera module, which chooses an active event for visualization based on the history of the narrative. They applied this technique to the first dialogue sequence of the game. In the original game, the player can only see it in the default first person view, however, their system turns it into a cinematographic scene. The system uses neural decider based on neural networks to classify actions in narrative events. Instead of creating a possibly large set of different hand-tuned, rule-based deciders, different neurons have been trained with examples of different action types and narrative events to make them classify the actions correctly. The system automatically creates more appealing cut-scenes for the game, but it cannot be used in an interactive context.

2.1.3.4 Automated Cinematography for Games

Cozic [1] looks at giving the camera system knowledge about what scene elements at a given moment in the game are important to the player. Each elements gets a shot contribution and if the avatar gets close enough to an important element the shot contribution will probability of how likely the element is to be included in the next camera shot. He is using atomic elements of cinematic principles to implement his method. One of the aims of the method is to always give the player a playable/usable or a 'fair' view as it is also described as. The idea of a 'fair' view is that the player should be able to play without risk of losing the game through an ill-positioned camera, for example the player cannot see platform which he/she is supposed to jump over to and thereby might risk not hitting the platform and die. The method is based on Hawkin's paper [22].

2.1.4 Summary

To summarise all of the constraint-based representations described above all benefit from being very flexible by allowing new constraints to be designed to create totally new visual properties. Recent work in particular [20] has shown their effectiveness in real-time. On the other hand the drawbacks of constraint-based systems are the high complexity of them. It requires an significant amount of time to get to understand and develop these systems. Most the existing work is slow (does not run in real-time) and the interface between the camera system and the artist/designer is complex. These drawbacks has been highly addressed in [20] which has tried to simplify the interface and optimise the search methods for real-time use. But [20] lacks in content knowledge which makes its camera system unable to know about important element to include, which is an issue which is addressed in [1]. One thing which can be improved in [1] is to better allow a smooth transition between showing important elements and showing what the player possibly intent to look at.

2.2 Camera Systems in Games

A camera system for a computer game decides what position and orientation of the camera in the game and thereby also decides how the player sees the game world. The player can have different degrees of freedom to control the camera while playing. Here has been experimented with many different ways of showing the action to the player. Below different example camera systems in games are reviewed:

2.2.1 Fixed Camera Systems

This type of camera system uses a set of hardcoded cameras positions and orientations. The camera may rotate or pan but the camera views will not change dramatically, so the same room or area will always be shown under the same set of views. When the avatar moves out of sight or is too far from the camera, the camera system switches to a new camera position and orientation. An example of a game which uses a fixed camera is *Metal Gear Solid Gear 2* as seen in figure 2.1.

One advantage of this camera system is that the game designer can fully use the camera as a rhetorical tool, and as a filmmaker, they have the option to create a mood through camerawork are careful selection of shots.

The downside of using this technique is that it is often difficult to make views thats both playable and dramatically relevant. In this type of camera system, the views should never prevent the player from seeing any important elements from the current camera view as the player will not be able to choose a better one. Figure 2.2 shows

examples of bad camera views from games using fixed camera.

Later games have started using a mix of fixed camera view and tracking views (described in next section).

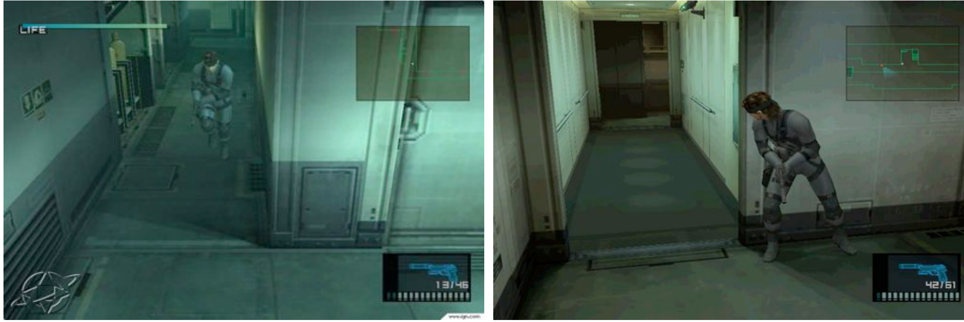
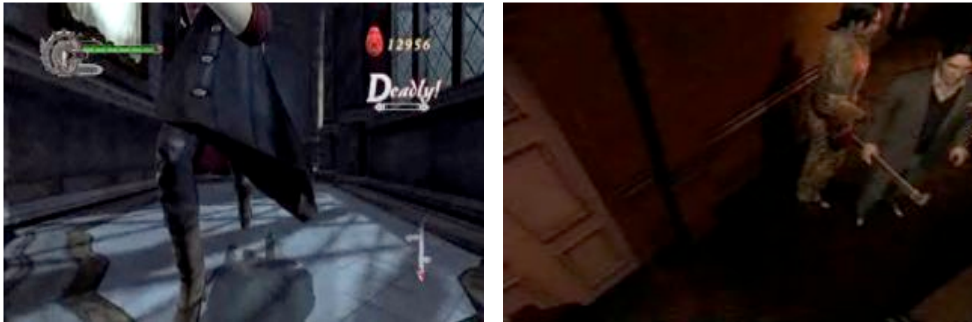


Figure 2.1: Screenshots from Metal Gear Solid 2 using fixed camera



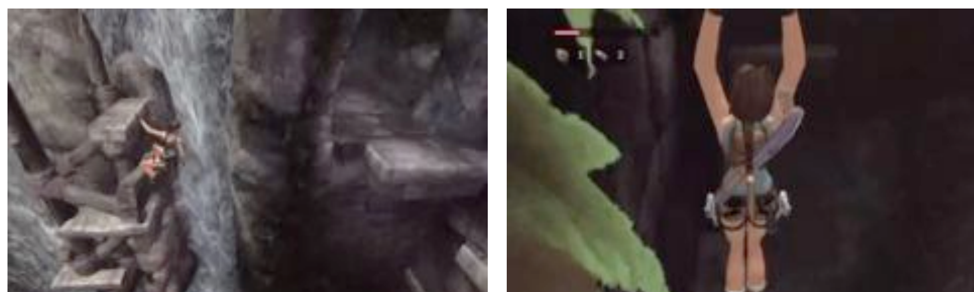
(a) Screenshot from Devil May Cry 4 (b) Screenshot from Resident Evil: Outbreak

Figure 2.2: Bad camera view from games using fixed camera

2.2.2 Tracking Camera Systems

As the name suggests this type of camera system simply follows the avatar from behind. This technique can cause problems if the player stands very close to a wall or when turning or it can be difficult for the player to judge distances for example if the player needs to make a jump to another platform. Early games such as *Crash Bandicoot* and *Tomb Raider* used very simple versions of the tracking camera system, and the player had no control over the camera and very often causes bad camera views for the player. More recent games are using more complex systems as for example spring systems and use ray casts techniques to check occlusions between the camera and the avatar to move

it to a more suitable position for example as seen in figure 2.3(a). Even more complex system bad camera views still occurs as seen in figure 2.3(b).



(a) Good camera view from Tomb Raider (b) Bad camera view from Tomb Raider

Figure 2.3: Screenshots from Tomb Raider

Some tracking camera systems also deal the camera views by allowing the player to partly control the camera for example by letting the player dolly or rotate the camera around the avatar. A classic example is seen in *Super Mario 64* where the player has the ability to rotate the camera by 45 degrees. This ability though creates problems in narrow spaces and the camera can easily become obstructed by objects in the scene.

2.2.3 Interactive Camera Systems

As described above some games offers semi-interactive camera to the player but there also games which offers full control of the camera. This has been done in for example *The Legend of Zelda: The Wind Waker* and *Super Mario Sunshine* where the player can both dolly and rotate the camera using a small joystick. At first sight this could be considered as a universal camera system as it could theoretically fit any situation. And yes it is always possible for the player to translate and rotate the camera to find a better view. However though it is normally good practice to keep the player's degrees of freedom as low as possible as the average player in general will lose concentration on gameplay itself when trying to concentrating on too parameters at once.

2.3 Choice of Camera System: Discussion

Currently none of the above systems with non-predefined views would be able to automatically reproduce views as seen in the examples with fixed camera. This is because the camera is only driven by the player's movement never by any further knowledge about the game world's elements such as story plot, character's emotions or impor-

tance to progressing in the game. As described above the fixed camera systems make use of cinematographic techniques but often hinders the player's actions by not showing him/her what he/she needs to see to be able to play.

Together with the above descriptions of the related works on camera control in games and the descriptions of the different types of camera system for games it has been chosen to focus on developing a camera system which is extends on the work by Cozic [1] using a modern tracking camera system like described in [20]. The reason for these choices is because the system in [1] does not only base it movement on the player's movements but it also uses metadata of the game world elements to check what elements to include in the camera view which fits with the aims of this thesis and mixed with the system from [20], a very efficient and easy-to-tweak camera system can be developed. Other reason for basing the camera system on [1] and [20] is that they both include a good description on how to implement their systems.

Chapter 3

Overview of Related AI Techniques

In this section different AI techniques for developing the decision maker method will be described and later compared to determine which of them is most suitable for developing the decision maker in this project. The descriptions are inspired by [23].

3.1 Finite State Machines

A finite state machine is a commonly used technique for letting an object behave differently in different situations. A state machine consists of four main elements:

- **States**, which define behaviour
- **State transitions**, which each are a movement from one state to another
- **Rules or conditions**, which must be met to allow a state transition
- **Input events**, which are either externally or internally generated and may possibly trigger rules

For example, a soldier could have the states *healthy*, *wounded* or *dead*. If he is in the state *healthy* he will engage the enemy solidiers in battle without thinking to much about avoiding fire. When his health drops below 50%, the rule for changing to state *wounded* will be met and the transition will occur. Now he will not blindly attack the enemy but will try to find cover to avoid enemy fire. If his health drops to 0% a transition to state *dead* will occur and he will no longer be able to shoot at the enemy.

In this way the behaviour of a soldier can be controlled. A problem with this technique is that it might result in very predictable behaviour. A solution could be to use nondeterministic state machines instead so that transitions between states are less predictable. For example, the soldier can change to state *wounded* if his health drops below 5o%, but if it drops below 60% there can be a small chance that he might already change state at this time. This could for example be determined by throwing a dice (random number).

The state machine could as such be used as the method for doing the decision making. For example each state could represent a specified view between the optimal and the player's view and the conditions could be how close or far the avatar is from an important element and how large or small the angle is between them. However, it can be hard to determine the exact conditions on when to make a transition from state to state and how many states to use, for example how big should the angle be to change to a new view and how many different views should there be.

Finite state machines was used in [15] but as mentioned in section 2.1.2.3 this technique does not work great in unpredictable virtual environments like computer games.

3.2 Genetic Algorithms

Genetic algorithms exploit the way evolution behaves in nature: Only the fittest members of a species survive. This ensures that only the DNA from the fittest members of a generation is passed on to the next generation, which results in the strongest possible generation being created. Furthermore, small errors will occur from time to time when passing on DNA from a parent to a child. This minor error might have some negative effect on the child, which will then have a smaller chance for passing its DNA on to the next generation. However, in some rare situations, these errors might result in positive effects for the child, i.e. making it fitter than all the other individuals from its generation and this might increase its chance for passing its DNA to a new generation.

In genetic algorithms different solutions to a problem are encoded for example in the form of a sequence of binary numbers. These solutions are then used to solve a problem and those that perform the best are selected for producing the next generation of solutions (and the worst are sometimes removed). This next generation is made by taking data from one solution and combining it with data from another solution. For example, if the solution is given by an array of ten binary numbers, a new child solution could be generated by taking the first five binary numbers from one solution and the last five from another. To ensure evolution, small changes will be made from time to time in this child. This could be by changing one bit from being '0' to being '1', or it could be to change the position of some bits. It is hoped that evolution will come up with a good solution to the problem after running several generations.

Genetic algorithms have been used with great success in relation to camera control. This is seen for example in [16] (described in more detail in section 2.1.2.4) but is not suited in a real-time application like a computer game, as this technique can only be used offline.

3.3 Neural Networks

Neural Networks are well known in the AI area and are described in many books dealing with AI for games. They can be defined as a model of reasoning based on the human brain. The model is made of many small building blocks called artificial neurons.

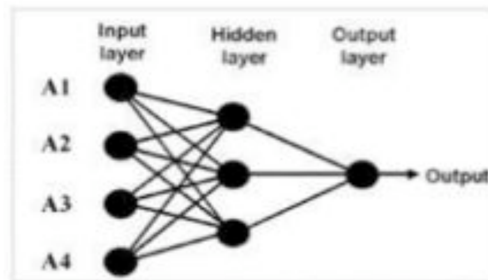


Figure 3.1: A feedforward neural network

An artificial neuron has an input and an output. Inputs normally reflect the state of the problem domain and the number of inputs to a neuron can vary a lot from network to network. Each input has a certain weight added to it. This weight specifies how much a particular input should be weighted inside the neuron. Within the neuron is an activation function. All the inputs are multiplied with their respective weights and are then added together. Different types of activation functions can be used for example a sigmoid function or just a simple step function (using for example 0 and 1 or -1 and 1). The output of the activation function is then sent on to other neurons or sent out of the network as one of the total outputs of the network.

The most widely used way to connect neurons is to use layers. The network in figure 3.1 is called a 'feedforward network' since each layer of neurons feeds its output into the next layer of neurons until an output is given. As seen from the figure, the feedforward network has a layer of neurons called the hidden layer. This layer is simply a layer of neurons lying between the inputs and the outputs. In fact, there can be as many hidden layers as one feels necessary or there can be no hidden layers at all.

A neural network can be trained for example to recognize a pattern. This is done by adjusting the weights in the network. A method for doing this is the 'back propagation' method. In this method, the output value of the network is compared to the target

output value. The difference between these two is called the error value. This error value is passed on to the output layer where the weights are adjusted according to the error value. This process then continues through the hidden layers by calculating the difference between the hidden layer next to the adjusted output layer. The weights in this layer are then adjusted and this continues until the input layer is reached.

A decision making method based on a neural network could be made by letting the input values be information about the distance between the avatar and an important element in the game world and the output could be an interpolation value of which view to view the most (the system's optimal view and the player's view).

One problem with using this method is that it is might seem like a black box. The box receives some inputs and it returns some outputs. However, to track down the relation between the inputs and the outputs becomes difficult because of all the different weights and connections between the neurons in the network, and therefore tweaking the method becomes a difficult task. Neural networks has also been used before in for example [21] (described in section 2.1.3.3) but this was in a non-interactive context.

3.4 Fuzzy Logic

A rule in classical logic is written in the form '*if A then B else C*'. Here the condition *A* must be exactly met if *B* is to be carried out. In other words *A* has to be specified in a way that makes it possible to clearly answer yes or no. If for example it is required to know whether a man is tall or not, it is necessary to clearly specify what tall implies. If the limit is set to being 180cm, then a person who is 179.99cm tall will not be seen as tall.

In fuzzy logic every statement can be seen as being true to some degree. If something is said to be true to a degree of 1 it is absolutely true while 0 means absolutely false [24]. Anything in between is true to some extent. For example, a person that is 160cm tall could be tall to a degree at around 0.8, while a person that is 210cm tall would tall to a degree of 1.

Fuzzy logic works by first converting an input value (a crisp value) into a degree of truth for a number of *fuzzy sets*. This process is called *fuzzification*. These sets could for example be short, medium and tall and will probably overlap each other a bit, which means that the input value can belong to one or more of the sets. A specific function for each set is used to fuzzify the crisp value. These functions are called membership functions and can vary widely from each other, which will affect the resulting degree of truth. Examples of membership functions are the triangular membership function, the curve membership function and the trapezoid membership function [24].

Once the degree of truth has been found for each of the sets, these degrees of truth are used to make the premise for fuzzy rules. The fuzzy rules are used to produce an

output that may remain fuzzy or – more commonly, especially in computer games – can be *defuzzified* to provide a crisp value. This is known as *fuzzy rule-based inference*, and is one of the most popular uses of fuzzy logic. See figure 3.2.

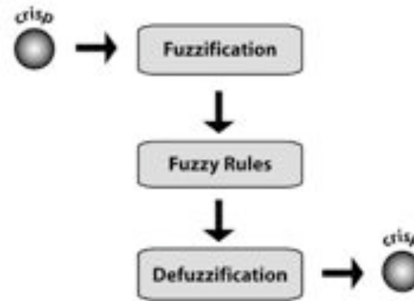


Figure 3.2: Fuzzy rule-based inference

Fuzzy expert systems can mimic human reasoning surprisingly closely, which makes them ideal for decision making, and therefore very suitable as a decision maker for the camera system. Fuzzy systems are also easy to extend incrementally. Each rule is modular, and different rules can be tweaked independently from others. One commonly cited problem with fuzzy systems is combinatorial explosion. Indeed this is true if all possible inputs combinations are to be enumerated as rules, but this is rarely necessary. So the only rules required are the ones that solve the problem, this is also known as the Combs method [25].

3.5 Choice of AI Technique: Discussion

The AI techniques described previously were finite state machines, decision trees, neural networks and fuzzy logic. There are two essential requirements needed to fulfill the aims of developing the system which are:

1. Character tracking
2. Object metadata knowledge

A fuzzy logic approach is suitable to use in such a system. Firstly, it is hard to tell exactly how close the avatar should be to an important element before alternating the player's view. Likewise, it is also difficult to determine how big the angle should be between the avatar and the element, before the camera system should change to another view. Fuzzy systems are ideal to use in mimicking human reasoning which is what is desired in this situation. Other advantages of fuzzy systems are that they are

very easy to manage and it is easy to tweak their individual rules, which is another aim for the decision maker. Finally, fuzzy logic is a well established AI method in game development.

Chapter 4

Fuzzy Logic Decision Maker

This chapter describes in detail the necessary theory of fuzzy logic and the fuzzy rules and fuzzy linguistic variables used for the fuzzy logic decision maker.

4.0.1 Fuzzy Sets

Crisp sets (a set where it is clearly defined what objects belongs to it or not) are useful but problematic in many situations. For instance, if the angle values in degrees from 0-180° are examined then the sets for $VerySmall$, $Small$, $Medium$, $Large$ and $VeryLarge$ could be defined as follows:

$VerySmall = \{0, 1, 2, ..., 29\}$

$Small = \{30, 31, 32, ..., 69\}$

$Medium = \{70, 71, 72, ..., 109\}$

$Large = \{110, 111, 112, ..., 149\}$

$VeryLarge = \{150, 151, 152, ..., 180\}$

A graphical way of showing these crisp sets is shown in figure 4.1. Note how the degree of membership of an element in any of the sets can be either 1 or 0.

Angles can now be categorized by assigning them to one of these sets based upon their angle value (in degrees). It is clear to see that an angle of 109 degrees is well above a medium sized angle and the majority of people would probably categorize it as a large angle. It is certainly much larger than an angle of 72 degrees even though both fall into the same category. It is also ridiculous to compare an angle of 69 degrees and an

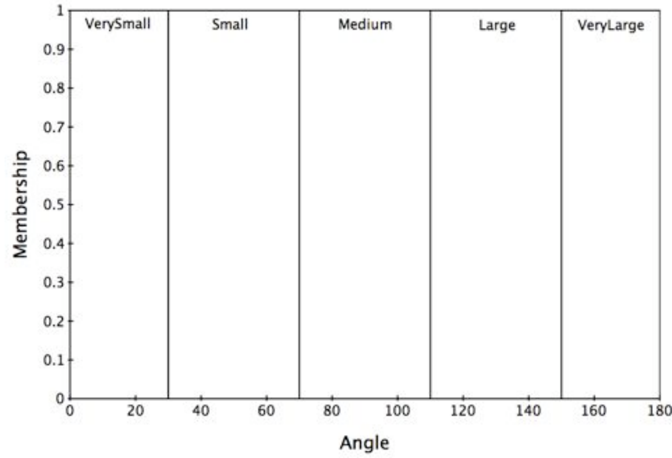


Figure 4.1: A graphical representation of a crisp set representing categories for angle values in degrees between 0-180°

angle of 70 degrees and come to the conclusion that one is small and the other is not. This is where crisp sets fall down. Fuzzy sets allow elements to be assigned to them to a matter of degree.

4.0.1.1 Membership functions

A fuzzy set is defined by a $membershipfunction$. These functions can be any arbitrary shape but are typically triangular or trapezoidal. Figure 4.2 shows a few examples.

The terms $VerySmall$, $Small$, $Medium$, $Large$ and $VeryLarge$, used from the crisp set example, are also referred to as linguistic terms. In figure 4.3 it is shown how these linguistic terms can be represented as fuzzy sets comprised of triangular membership functions. The dotted line shows how an angle of 115 degrees is a member of two sets. The angle's degree of membership in $Large$ is 0.625 and 0.375 in $Medium$.

4.0.1.2 Fuzzy Set Operators

Intersections, unions and complements of fuzzy sets are possible, just as they are with crisp sets. The fuzzy intersection operation is mathematically equivalent to the AND operator. The AND operator is equivalent of taking the minimum DOM (degree of membership) for each set that a value is a member of. Written like this:

$$C = A \cap B \Leftrightarrow C(x) = min(A(x), B(x)), \forall x \tag{4.1}$$

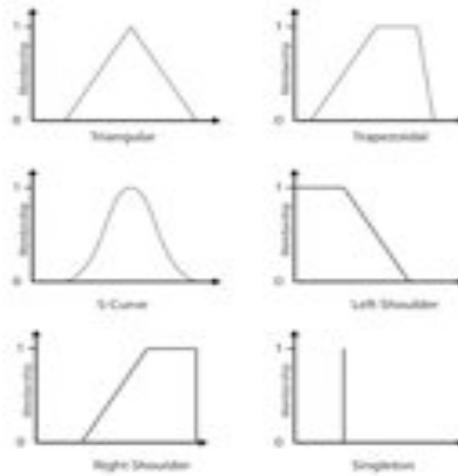


Figure 4.2: Examples of member functions for fuzzy sets

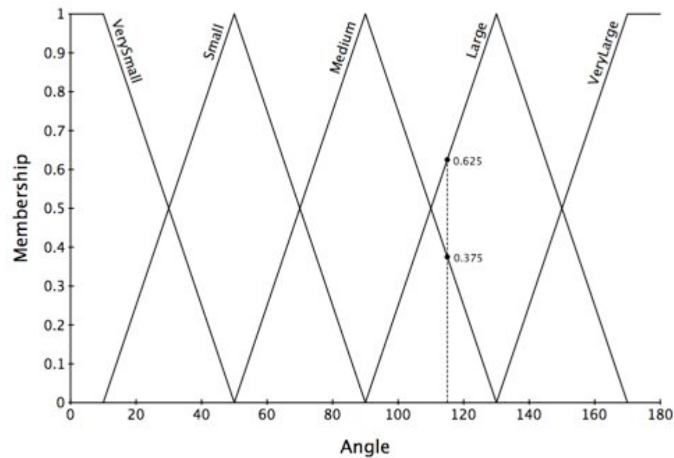


Figure 4.3: Dumb, Average, and Clever as fuzzy sets. The dotted line represents an angle of 115 degrees, and its points of intersection with the sets $Large$ and $Medium$ represent its degree of membership in those sets.

The union of fuzzy sets is equivalent to the OR operator, written like:

$$C = A \cup B \Leftrightarrow C(x) = max(A(x), B(x)), \forall x \tag{4.2}$$

And the complement of a value with a DOM of m is 1-m, written like this:

$$\neg A(x) = 1 - A(x) \tag{4.3}$$

4.0.2 Fuzzy Linguistic Variables

A *fuzzy linguistic variable* (or FLV) is the composition of one or more fuzzy sets which represent a concept or domain qualitatively. Given the earlier example, the sets $VerySmall$, $Small$, $Medium$, $Large$ and $VeryLarge$ are members of the FLV **Angle**. This can be written in set notation as:

$$\textbf{Angle} = \{VerySmall, Small, Medium, Large, VeryLarge\}$$

4.0.3 Fuzzy Rules

Fuzzy rules are comprised of an *antecedent* and a *consequent* in the form:

IF *antecedent* THEN *consequent*

The antecedent represents a condition and the consequent describes the consequence if that condition is satisfied. The form is similar to conventional rules, but unlike the conventional rules where the consequent either fires or not, the consequent can fire to a matter of degree in fuzzy systems.

The antecedent can be a single fuzzy term or the set that is the result of a combination or several terms. The degree of membership of the antecedent defines the degree to which the consequent fires. A fuzzy inference system is typically comprised of many such rules, the number of which is proportional to the number of FLVs required for the problem domain and the number of membership sets those FLVs contain. Each time a fuzzy system iterates through its rule set, it combines the consequents that have fired and defuzzifies the result to give a crisp value.

4.0.3.1 Fuzzy Inference

The procedure of fuzzy inference is where the fuzzy system is presented with some values, which it will use to check which rules fire and to what degree. Fuzzy inference follows these steps:

1. For each rule,

    (a) For each antecedent, calculate the degree of membership of the input data.

   (b) Calculate the rule's inferred conclusion based upon the values determined in 1a.

2. Combine all the inferred conclusions into a single conclusion (a fuzzy set).

3. For crisp values, the conclusion from 2 must be defuzzified.

4.0.3.2 Defuzzification

$Defuzzification$ is the reverse of fuzzification: the process of turning a fuzzy set into a crisp value. There are many techniques for doing this and this section will examine the most common techniques.

Mean of Maximum (MOM)
The mean of maximum – MOM for short – method of defuzzification calculates the average of those output values that have the highest confidence degrees. One problem with this method is that it does not take into account those sets that do not have confidences equal to the highest, which can bias the resultant crisp value to one end of the domain. More accurate defuzzification methods such as the ones described below resolve this problem.

Centroid
The centroid method is the most accurate method but is also the most complex to calculate. It works by determining the center of mass of the output sets. If you imagine each member set of the output set cut out of card stock and then glued together to form the shape of the fuzzy manifold, the center of mass is the position where the resultant shape would balance if placed on a ruler.

The centroid of a fuzzy manifold is calculated by slicing up the manifold into s sample points and calculating the sum of the contribution of the DOM at each sample point to the total, divided by the sum of the DOMs of the samples. The formula look like this:

$$CrispValue = \frac{\sum_{s=DomainMin}^{s=DomainMax} s \times DOM(s)}{\sum_{s=DomainMin}^{s=DomainMax} DOM(s)} \tag{4.4}$$

where s is the value at each sample point and DOM(s) is the degree of membership in the FLV of that value. The more sample points used to do the calculation, the more accurate the result gets, although in practice 10 to 20 samples are usually enough.

Average of Maxima (MaxAv)
The maximum or *representative value* of a fuzzy set is the value where membership in that set is 1. For triangular sets this is simply the value at the peak, for sets containing plateaus – such as right shoulder, left shoulder and trapeziods sets – this value is the average of the values at the beginning and end of the plateau. The *average of maxima* (MaxAv for short) defuzzification method scales the representative value of each consequent by its confidence and takes the average, like so:

$$CrispValue = \frac{\sum representativevalue \times confidence}{\sum confidence} \tag{4.5}$$

This method can produce values that are very close to values (for the same defuzzification) calculated by the more accurate but costlier centroid technique and therefore this method is suitable for games where computation cost is a concern.

4.0.4 The Combs Method

One major problem with fuzzy inference systems is that as the complexity of the problem increases, the number of rules required escalates at an exponential rate. The Combs method [27] allows the number of rules to grow linearly with the number of member sets instead of exponentially. Table 5.1 shows the number of rules required using the traditional method versus the Combs method (assuming each FLV contains five member set).

Table 4.1: Rules Required (traditional) vs Rules Required (Combs)

Number of FLVs	Rules Required (traditional)	Rules Required (Combs)
2	25	10
3	125	15
4	625	20
5	3,125	25
6	15,625	30
7	78,125	35
8	390,625	40

The theory behind the Combs method works on the principle that a rule such as:

IF *Target_Far* AND *Ammo_Loads* THEN *Desirable*

is logically equivalent to:

IF *Target_Far* THEN *Desirable*

OR

IF *Ammo_Loads* THEN *Desirable*

One of the drawbacks with this method is that the changes to the rule base required to accommodate the logic are not intuitive.

4.1 Fuzzy Logic Setup in FLCam

This section describes the details of the setup of the fuzzy logic for the decision maker. This includes details on the FLVs and the fuzzy rules.

4.1.1 FLVs

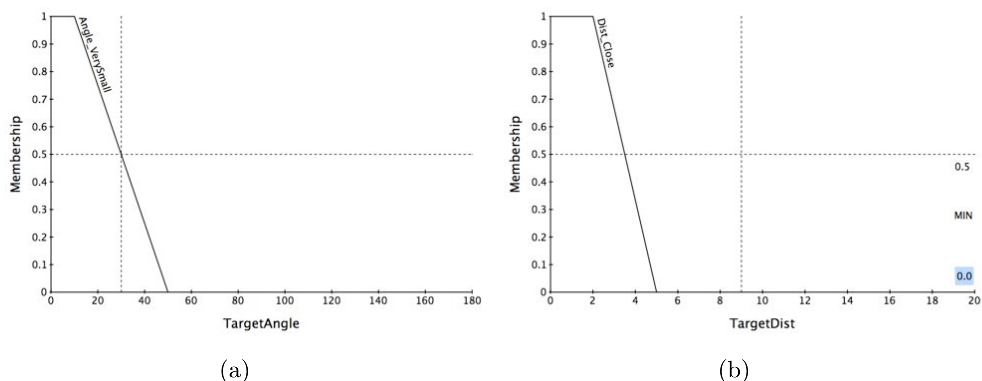
Please see appendix A for the graphical representations of the FLVs with their membership functions used for the decision maker in FLCam.

4.1.2 Fuzzy Rules

This section describes an example scenario which goes through some of the rules to let the reader get a better understanding of what is going on in the fuzzy system. In the scenario the angle between the avatar and the look-at point is 30 degrees, the avatar will have a distance of 9 units away from the current look-at point and the average shot contribution from the current shot is 0.4 (the shot contribution will only affect the last three rules for the shot weight). The following paragraphs goes through some of the first rules (see appendix A with details of all the fuzzy rules used in FLCam):

Rule One: IF *Angle_VerySmall* **AND** *Dist_Close* **THEN** *SW_Low*

The degree of membership of the value 30 in the set *Angle_VerySmall* is 0.5. The degree of membership of the value 9 in the set *Dist_Close* is 0. The AND operator results in the minimum of these values so the inferred conclusion for Rule 1 is *SW_Low*=0. In other words, the rule does not fire. Figure 4.4 shows this rule graphically.

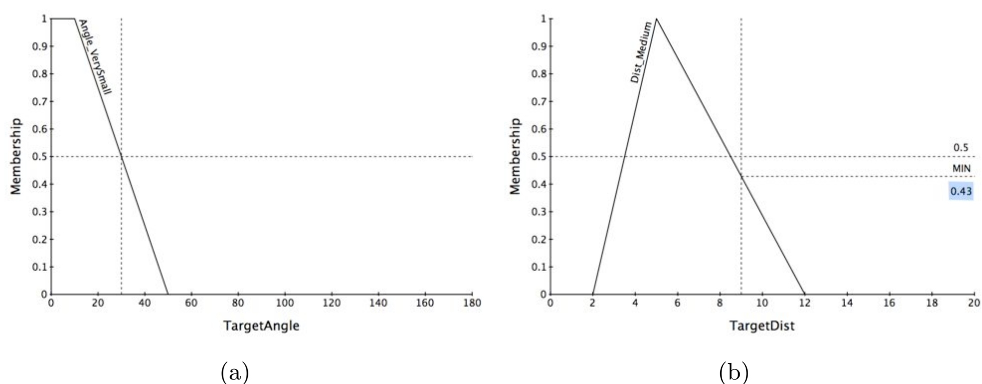


(a) (b)

Figure 4.4: (a) shows the degree of membership of the value 30 to the set *Angle_VerySmall* and (b) shows the degree of membership of the value 9 in the set *Dist_Close*

Rule Two: IF *Angle_VerySmall* **AND** *Dist_Medium* **THEN** *SW_Medium*

For the second rule, the degree of membership of the value 30 in the set *Angle_VerySmall* is 0.5. The degree of membership of the value 9 in the set *Dist_Medium* is 0.43. The inferred conclusion for Rule 2 therefore is *SW_Medium*=0.43. See figure 4.5.

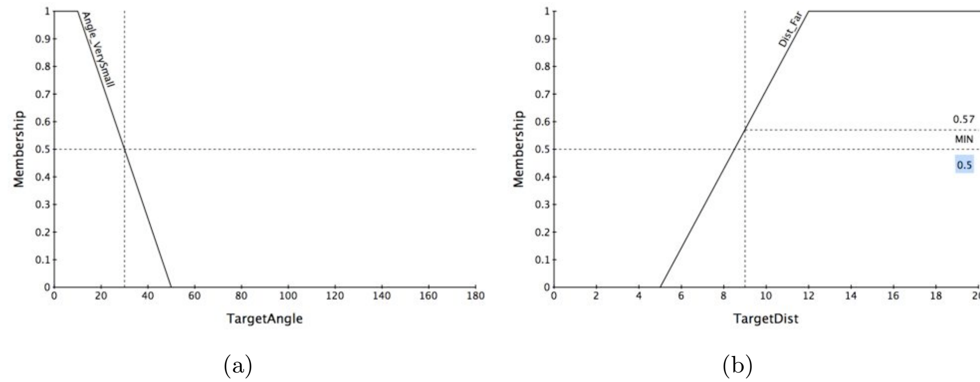


(a) (b)

Figure 4.5: (a) shows the degree of membership of the value 30 to the set *Angle_VerySmall* and (b) shows the degree of membership of the value 9 in the set *Dist_Medium*

Rule Three: **IF** *Angle_VerySmall* **AND** *Dist_Far* **THEN** *SW_High*

As a final example, the third rule, the degree of membership of the value 30 in the set *Angle_VerySmall* is 0.5. The degree of membership of the value 9 in the set *Dist_Far* is 0.57. The inferred conclusion for Rule 2 therefore is *SW_High*=0.5. See figure 4.6.



(a) (b)

Figure 4.6: (a) shows the degree of membership of the value 30 to the set *Angle_VerySmall* and (b) shows the degree of membership of the value 9 in the set *Dist_Far*

The same method goes for the rest of the rules to find the final crisp shot weight value. The inferred results for all the rules are summarized by the matrix shown in table 4.7 (this type of matrix is known as a *fuzzy associative matrix*, or FAM for short).

Note that both *SW_Medium* and *SW_High* has fired twice but *SW_Low* only once. One way to think of these values is as confidence levels. Given the input data, the fuzzy rules have inferred the result *SW_Low* with a confidence of 0.25. But what conclusion is inferred for *SW_Medium* and *SW_High* since they have fired twice each? Well, there are a few ways of handling multiple confidences. The two most popular are bounded sum (sum and bound to 1) and maximum value (equivalent to ORing the confidences together). The maximum value method has been chosen for this project.

To summarize, table 4.2 lists the inferred conclusions of applying the values of angle between avatar and look-at point = 30, distance between avatar and look-at point = 9 and average shot contribution = 0.4 to all the rules.

The results are shown graphically in figure 4.8. Notice how the membership function of each consequent is clipped to the level of confidence.

	Angle_VerySmall	Angle_Small	Angle_Medium	Angle_Large	Angle_VeryLarge
Dist_Close	SW_Low 0	SW_Low 0	SW_Low 0	SW_Low 0	SW_Low 0
Dist_Medium	SW_Medium 0.43	SW_Medium 0.43	SW_Low 0	SW_Low 0	SW_Low 0
Dist_Far	SW_High 0.5	SW_High 0.5	SW_Low 0	SW_Low 0	SW_Low 0

(a)

Contrib_Low	Contrib_Medium	Contrib_High
SW_Low 0.25	SW_Medium 0.75	SW_High 0

(b)

Figure 4.7: The FAM for the shot weight rule base given the input values target angle = 30, target distance = 9 and the average shot contribution = 0.4. The shaded cells highlight rules that have been fired.

Table 4.2: Summary of example

Consequent	Confidence
SW_Low	0.25
SW_Medium	0.75
SW_High	0.5

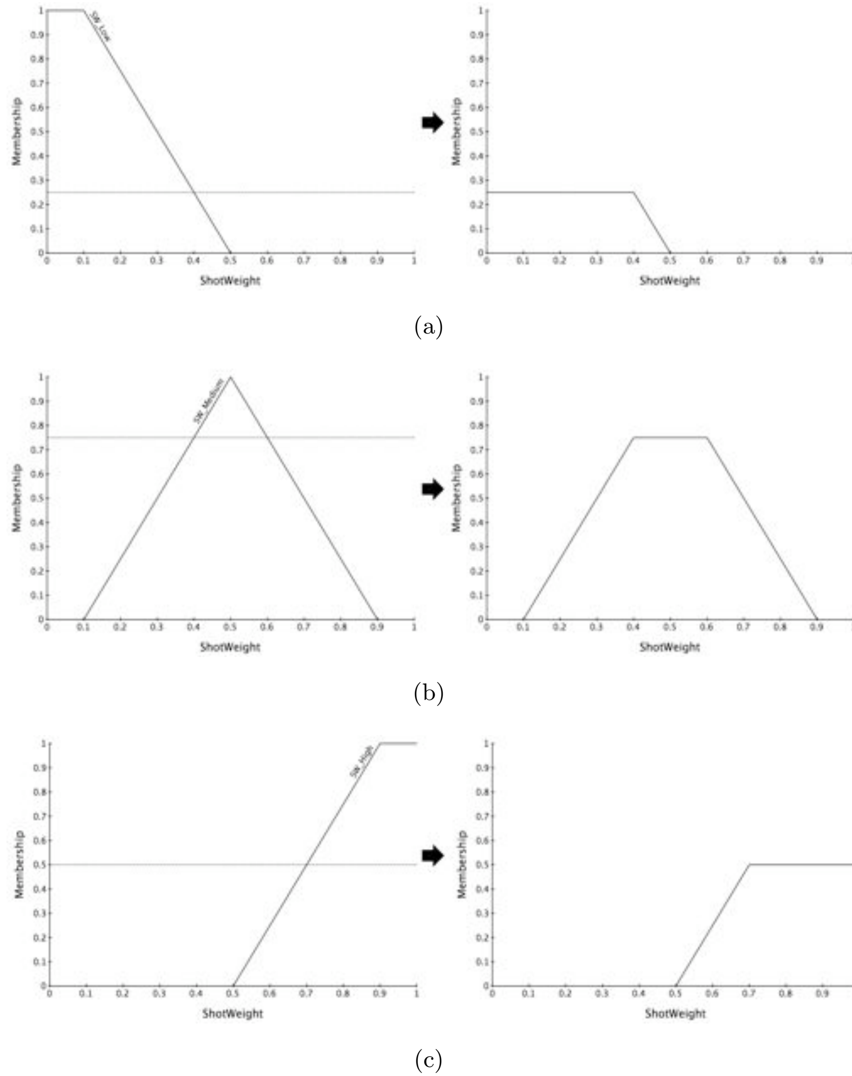


Figure 4.8: The inferred results of processing the rule set for shot weight

The next step is to combine the inferred results into a single fuzzy manifold. See figure 4.9.

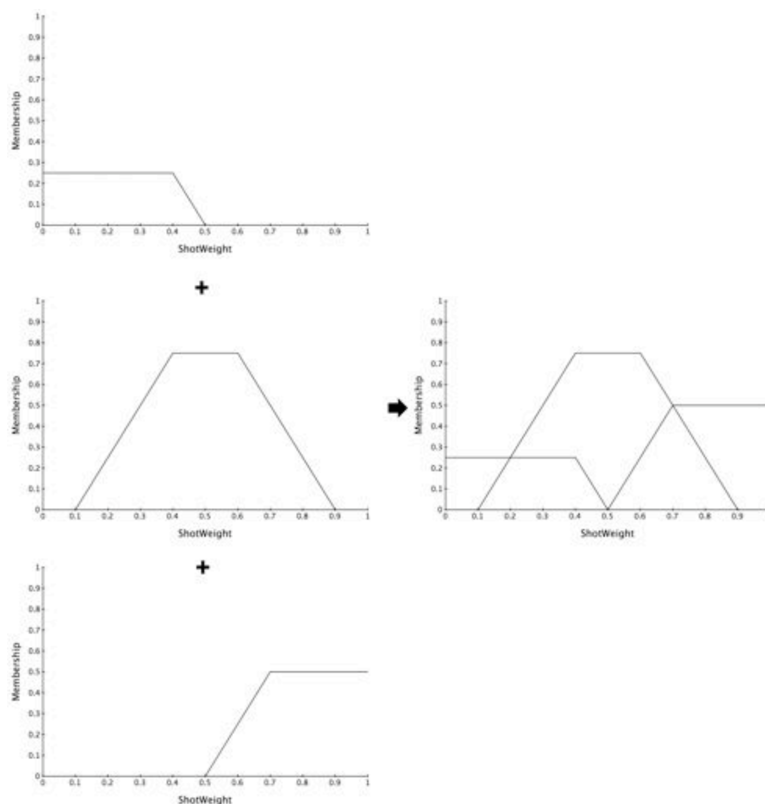


Figure 4.9: Combining the conclusions

After combining the conclusions the last step is the defuzzification. As mentioned earlier this project will be using the Average of Maxima defuzzification method (see section 4.0.3.2). To recap the following equation is used:

$$CrispValue = \frac{\sum representativevalue \times confidence}{\sum confidence}$$

The representative values of the sets comprising the output manifold are summarized in table 4.3.

Plugging these values into the equation gives the shot weight as a crisp value:

Table 4.3: Representative values of the sets comprising the output manifold

Set	Representative Value	Confidence
SW_Low	0.2	0.25
SW_Medium	0.5	0.75
SW_High	0.85	0.5

$$\begin{aligned} ShotWeight &= \frac{0.2 \times 0.25 + 0.5 \times 0.75 + 0.85 \times 0.5}{0.25 + 0.75 + 0.5} \\ &= \frac{0.85}{1.5} \\ ShotWeight &\approx 0.57 \end{aligned} \tag{4.6}$$

The example above has gone from crisp values (angle between avatar and look-at point = 30, distance between avatar and look-at point = 9 and average shot contribution = 0.4) to fuzzy sets, to inference and back to a crisp value representing the shot weight (or interpolation value) between the system's optimal view and the player's intentional view. This is the technique used to smoothly find a view between these two views. Please see figure 4.10 for the graphical outputs for shot weight 0.2, 0.5, 0.85 and 0.57 respectively.

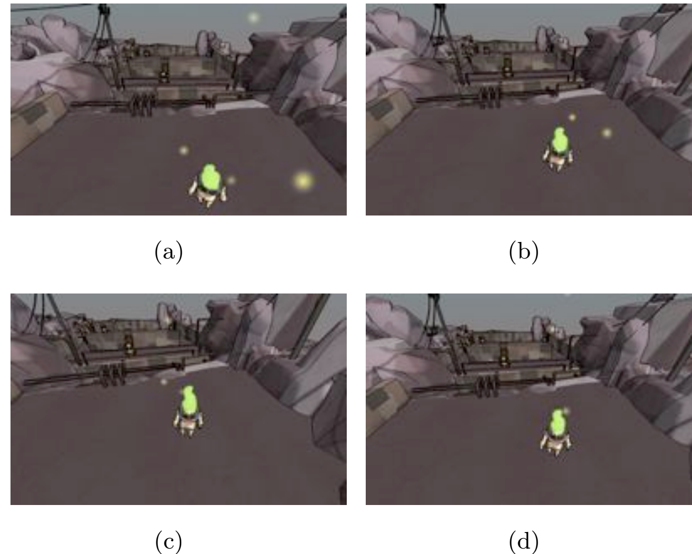


Figure 4.10: The graphical outputs of shot weight 0.2 (a), 0.5 (b), 0.85 (c) and 0.57 (d)

Chapter 5

The Implementation on the Game Engine

This chapter presents the implementation details of the camera system and the fuzzy logic decision maker. Furthermore it discusses the motivation for whether to integrate the camera system (and decision maker method) in an already existing game or whether to construct a new game from scratch. It shows the main steps necessary for the integration, and gives an evaluation of the results.

5.1 Use an Existing Game or Create a New One?

To be able to test the performance of the decision making method implemented in this project, the method needs to be integrated in some type of computer game. This game could either be an existing game with accessible source code, or it could be a small prototype game developed specifically for this project only.

Many of the related works described in chapter 2.1 have created their own small games or applications to test their camera systems. The reason for this is most likely because they did not have access to a suitable game with including source code and when you create your own game, you can create exactly what you need to test your methods and systems. However, the disadvantage of creating a "game" that just tests what you need is that you do not get a feel of how it would work in connection with a fully functional game. There can be many different aspects which can overseen when just evaluating your system in a small controlled world.

It has therefore been chosen to use an already implemented game, more specifically the game Puzzle Bloom [26]. The main reason for choosing this game as the test

environment is that it has various puzzles throughout the game, where camera hints could be very useful to unexperienced players in aiding them through the game.

Puzzle Bloom (figure 5.1) can be described as a type of adventure game. The player needs to use the avatar, a small green fairy, to control the creatures in each level and move them around. The player has to solve small puzzles during each level by moving the creatures and placing them on buttons which open doors to allow other creatures to walk through them and thereby progressing in the game. The game is implemented in the Unity game engine which automatically makes the choice of what game engine to use.

Figure 5.1: Screenshot from Puzzle Bloom, the game which will be used as the test bed

Different elements become important to the player as the player progresses in the game. Some players might not totally figure out how to solve some of the puzzles and this is exactly the kind of situation where the decision maker can be tested, when exploring the space to select the best camera view for the camera hints.

5.2 General System Structure

One of the main ideas behind this work was to develop a camera system that can be used in almost any kind of games providing metadata about the objects in the game world. Motivated by fuzzy logic's ability to closely mimic human reasoning, a decision maker module has been created based on a fuzzy system, which can easily be implemented in an existing game and can quickly be extended and tweaked if necessary. To create a camera system that is independent of specific knowledge of a game object, an interface is implemented for all game objects (described later) and used by the camera system for getting the metadata from the objects. The camera system receives the object metadata through *scene object events*.

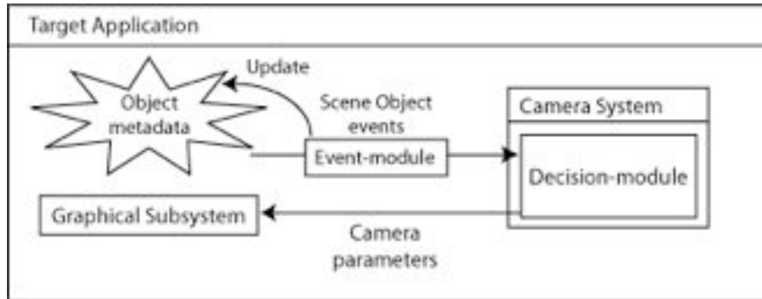


Figure 5.2: General system structure and flow of information

A sketch of the overall system and the flow of information between the camera system and the game application is shown in figure 5.2. An event-module sends metadata of player-relevant objects in the form of story/progress events to the camera system. Based on how far and at what angle the avatar is from the relevant objects, the decision-module chooses a suitable set of camera parameters (position and orientation) and sends them to the graphical subsystem of the target application. During some events specified by the game designer, objects can get their metadata updated, for example if the player presses a button which opens a door at another location, then the button is not important anymore but the door is now important (but it was not before the button was pressed).

5.3 The Algorithm

This is the algorithm used for the implemented camera system. The parts of the algorithm will be described in details in the following sections.

1. Scene Object Events
   (a) Check if any new important objects nearby
2. Shot contribution update
   (a) Decay shot contributions for scene objects, remove old objects
3. Current shot update
   (a) Check if a new shot should be chosen
      i. Apply object rules
      ii. Choose the highest prioritized shot as the current shot

4. Camera parameters setup
   (a) When scene objects are in the current shot
      i. Calculate the look-at point from the current shots scene objects
      ii. Calculate the radius
      iii. Calculate distance, height and orientation weights using fuzzy logic
      iv. Calculate shot weight (interpolation value) using fuzzy logic
      v. Calculate ideal camera position without scene objects included
      vi. Setup the line of action for the current shot (including objects)
      vii. Calculate ideal camera position with scene objects included
      viii. Using the shot weight interpolate between the look-at points with and without including the current shot's scene objects
      ix. Using the shot weight interpolate between the direction of the line of action for the current shot and the avatar's forward vector
      x. Using the shot weight interpolate between the radius with the current shot's scene objects and the minimum allowed radius
   (b) Else
      i. Set look-at point to the avatar's position
      ii. Set the radius to the avatar's full shot radius
      iii. Setup follow behaviour for the avatar only
5. Constraint Solver
   (a) Check if the avatar is visible
   (b) Check if the scene objects in the current shot which are supposed to be visible also are visible
   (c) If not
      i. Try a new alternative position close the initial position
      ii. Repeat until all constraints are satisfied for maximum n iterations
6. Calculate frame coherence using fuzzy logic
7. Interpolate between old and new camera position and rotation

5.4 Scene Object Events

During every update step, events can occur to let previously unimportant objects become important. When this happens and the player gets close enough to one of the objects, the object will be added by the event module to the list of scene objects (SOs) in the camera system, which means it has the possibility to be included in the next current shot. See figure 5.3 for a graphical representation of the scene object events.

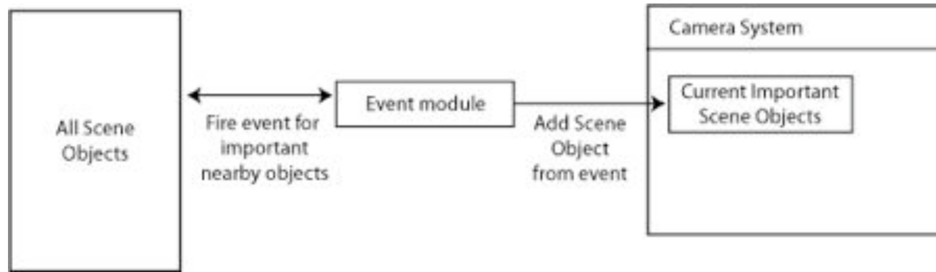


Figure 5.3: Scene Object Events

foreach *SceneObject so in allSceneObjects* **do**
  if *so.shotContribution > 0* **then**
    visible = is visible to avatar OR so.alwaysVisible;
    angle = angle between avatar and so;
    if *visible AND angle < ignoreAngle* **then**
      fire scene object event;
    end
  end
end

Algorithm 1: The algorithm used for firing scene object events

The following algorithm is used for firing events:

As algorithm 1 shows, a scene object event will only be fired if a line between the scene object and the avatar can be drawn with obstacles in between the line (unless special case is specified for the object) and the angle between the avatar and the scene object is less than some user-defined angle. The ignore angle is used so the player can walk away from the object without being annoyed by the camera system trying to include the object again. For more details see [1].

5.5 Shot Contribution Update

Each SO has a shot contribution (SC) value between 0 and 1. If the SO is part of the list (an event has occured to include it), the object's SC value will decay during every update at some decay rate assigned to this object. When the value reaches 0 the SO will be removed from the list, and hence will not be included in any future shots unless it is added to the list again. For more details see [1], as this is done in a similar fashion in Cozic's work.

5.6 Current Shot Update

Each shot is set to last for a minimum period of time to avoid the camera jumping too frequently. When the minimum period of time has past, the camera system will go through a set of specified rules to check what objects can be included in the next shot. The shot with the highest priority will be chosen as the next shot. If no objects are to be included, the default shot is chosen, which is basically a shot were just the avatar is included. See appendix B for more details on the object rules. The format of the rules are inspired by the format from [1].

5.7 Camera Parameters Setup

This section describes when one or more SOs are included in the current shot. When no SOs are included the ideal camera position is calculated in a similar way without using the fuzzy logic decision maker.

5.7.1 Camera Shot Setup

During each update step, the system uses the following algorithm to calculate the camera parameters and thereby setup the camera shot:

First the average position (look-at point) of the SOs for the current shot are calculated. Hereafter, the furthest distance between any two of the SOs is calculated, which will be used as the radius later on. An equation based on the equation from [22] is used to find the distance the camera should be away from the look-at point to ensure all the SOs are included in the shot. It looks like this:

$$d = (UnitPlaneLength(e) * size)/(screenSize * Tan(fieldOfView/2)) \tag{5.1}$$

Where UnitPlaneLength(e) is basically equal to 1 if the emphasis (e) is full (can also be two-thirds or one-half, see [22] for more details), $size$ is the radius in this case, $screenSize$ indicates what type of shot is used for the object where 1 indicates a full-shot and 0 a close-up shot. 1 was used for $screenSize$ to create a full shot of the "virtual" created sphere with the look-at point as its center and the calculated radius as its radius. And last $fieldOfView$ is the camera's vertical field of view. See figure 5.4 for an example.

The constraints used in [20] are *height*, *distance* (in the ground-plane) and *orientation* and will also be used in this context. The orientation is set up to be along the line of action which is defined to be in the direction from the avatar to the look-at point. The

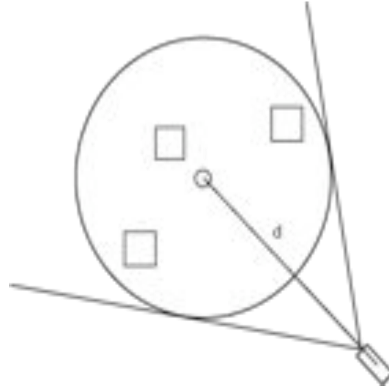


Figure 5.4: Top-down view example of how the camera system sets up the camera shots including scene objects

last two constraints are calculated using the desired angle θ between the ground plane and the direction from look-at point to the camera's position:

$$h = Sin(\theta) \cdot d$$
$$d_{xz} = Cos(\theta) \cdot d$$

These constraints use weights to allow the game designer to create profiles for the camera's behaviour. The higher the weight values the faster the camera will reach the ideal values for the distance, height and orientation respectively. When SOs are included, the fuzzy logic decision maker chooses what weight values to use which will be described later. When no SOs are included, the weights have been assigned as follows:

$$distanceWeight = 0.5$$
$$heightWeight = 0.5$$
$$orientationWeight = 0.8$$

The ideal position for the camera can be found using these three constraints and its rotation will be set to look at the look-at point.

5.7.2 Setup of the line of action

The line of action is only used when SOs are included in the current shot. It is calculated the following way:

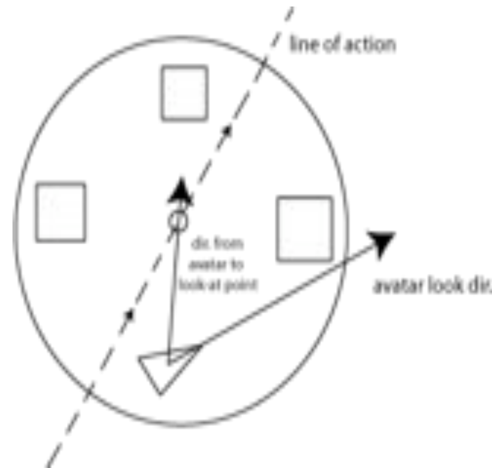


Figure 5.5: Example of how the line of action is calculated when SOs are present in the current shot

First the direction of the line of action is calculated by interpolating between the direction from the avatar's position to the look-at point and the avatar's look direction (its forward vector). The value 0.1 was used during the testing as the interpolation value between the two vectors (0 is completely look-at dir. vector and 1 is completely avatar's look dir.). The line of action will then be set to go through the look-at point.

The line of action is then used for when setting up the orientation for the camera. For example if the camera should be looking in the same direction as the line of action, the orientation value would then be 0 degrees and in that way it can be specified the camera should be relative to the line of action.

5.7.3 Choosing Camera Behaviour Using Fuzzy Logic

As mentioned earlier the fuzzy decision maker is used when SOs are included in the current shot to choose the values for the distance, height and orientation weights. The reason for doing this is to create dynamic camera behaviour for when moving the camera around when SOs are included in the shot. For example if the avatar gets close to a SO, it is necessary to draw attention to this SO and move the camera quickly towards this area. The change in camera behaviour will most likely also notify the player that something is happening and there is a big chance the player will take notice of the SO in the shot. The fuzzy rules for camera behaviour are described in section 4.1.2.

5.7.4 Choosing Shot Weight Using Fuzzy Logic

To decide what camera view to use when SOs are included in the current shot, the camera system uses two different camera setups (position and rotation) to interpolate between, namely the ideal camera position when the SOs are included in the shot and the ideal camera position for when just the avatar is included. The technique described in section 5.7.1 is used in both cases to calculate the ideal position.

When calculating the ideal camera position and rotation when the SOs are ignored, the look-at point is set to be the centroid of the avatar and the radius is set at half its height (a sphere with that radius and center at the centroid can cover the whole avatar). The camera system ends up with two sets of positions and rotations for the camera, one with the SOs and the other without. The next step is where the fuzzy logic decision maker comes in.

The fuzzy system (rules described in section 4.1.2) is used to calculate a so-called shot weight value, which basically is used as an interpolation value to interpolate between the look-at points, the directions of the lines of action and the radii from the two newly found camera setups as described above. Figure 5.6 shows the resulting outcome from an example situation.

Predicted future avatar position

A method predicting the avatar's position has been developed to take the player's future movements into account when making decisions on what camera views to use. The formula used for predicting the avatar's future position is shown here (for more details see [28]):

$$P(t+u) = P(t) + \frac{s(t)}{a(t)}(D(t)sin(u \cdot a(t)) + R(t)(1 - cos(u \cdot a(t)))) \quad (5.2)$$

where P(t) is the position of the avatar at time t, V(t) is the avatar's estimated velocity, s(t) is the avatar's speed at time t, D(t) is the heading direction of the avatar at time t, a(t) is the turning rate of the avatar at time t, R(t) is the right vector of D(t) and P(t+u) is the avatar predicted position at time t+u. The avatar's angular speed has also been taken into account as the naive approach of just adding the current velocity multiplied by the number of seconds in the future which can easily be broken if the player just runs around in tight circles. The calculated future positions are also interpolated with the current position to not create future positions which jump too much. During the tests u was set to 30 frames.

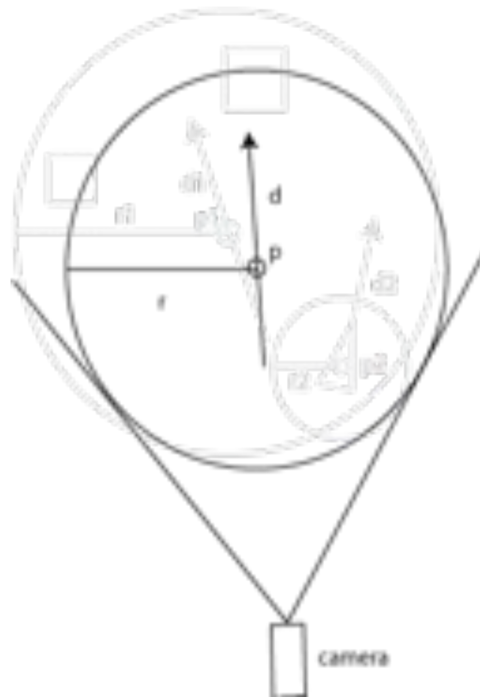


Figure 5.6: An example situation of how the fuzzy decision maker uses the look-at points (p1 and p2), the direction vectors from the lines of action (d1 and d2) and the radii (r1 and r2) from the two camera setups with and without the scene objects to find the new camera setup (p, d and r)

5.7.5 Constraint Solver

The constraint solver includes as many of the required SOs as possible in the next shot. Firstly, it checks if all the SOs and avatar are visible from the current camera position. If this test fails (i.e. at least one object is not visible) a new camera position will be found. If it is not possible at all to include all the SOs in the same shot, the least important object (the one with the lowest SC) will be removed from the current shot. The constraint solver uses the following algorithm to find a suitable camera position

and orientation:

```
i = 0;
n = MAX_ITERATIONS;
a = default shot angle index;
objectsReomved = 0;
while SOs and avatar are not visible do
    r = 0;
    step = 180 / (n/2);
    c = i % 2 == 0 ? -1 : 1;
    cc = i * step;
    if i % 2 != 0 then
        cc = i * step - (i/2) * step;
    end
    r = c * cc;
    Setup camera parameters for rotation r and shot angle angles[a];
    i++;
    if i > n then
        i = 0;
        a++; if a >= angles.Length then
            a = default shot angle index;
        end
        if a == default shot angle index then
            Remove the least important SO;
            objectsReomved++;
            if objectsReomved == MAX_OBJ_REMOVALS then
                stop;
            end
        end
    end
end
```

Algorithm 2: The constraint solver algorithm

Here the rotation r is used relative to the line of action, so if the $r=0$ then the camera will be completely aligned in the same direction as the line of action and negative and positive values will bring the camera to either side of the line of action.

5.7.6 Calculate Frame Coherence Using Fuzzy Logic

To create a smooth transition between two look-at points, a few fuzzy rules has been created to calculate different frame coherence for different situations. If there is a big jump between two look-at points, the camera's orientation quickly interpolates to the new look-at point which can create a very jumpy feeling. So the fuzzy rules makes it possible to slow down the interpolation of the two different camera orientations, which

would normally be quicker.

5.7.7 Interpolation Between the Old and New Camera Setup

The last step is simply to spherical interpolate between the old and new camera position and orientation. The calculated frame coherence is used for the interpolation of the orientations. The position interpolations use a constant frame coherence.

5.8 Scene Object Interface

Every object in the game world, which needs to be regarded by the camera system, must implement the Scene Object Interface which will be described in this section.

Table 5.1: Scene Object Interface

Name	Type
ImportantThroughWalls	Boolean
MustBeVisible	Boolean
ObjectCenter	Vector3
FullShotRadius	Float
ShotContribution	Float
SCDecayRate	Float
InfluenceDistance	Float

ImportantThroughWalls is used to let the camera system know if the SO should be included in a shot even if there is an obstacle in between the avatar and the SO. *MustBeVisible* indicates whether the camera should be adjusted to make the SO visible or if it can be ignored by the system. *ObjectCenter* is the position the camera system should use as the SO's position. This position should be the center of the SO's bounding sphere and *FullShotRadius* is the bounding sphere's radius, which is also the radius used by the camera system to set up a full shot of the SO. *ShotContribution* would normally be set to 0 from the beginning of the game then later activated to 1 for example during some kind of event in the game which tells the system that the SO has become important to the player, but *ShotContribution* can also be initially 1 if it is important to the player from the moment the game starts. *SCDecayRate* specifies how quickly the SC should decay i.e. how quickly the SO should become unimportant to the player. *InfluenceDistance* specifies how close the player has to be from the SO before the camera system will start including it in the camera shots.

Chapter 6

Example Scenarios

This chapter will go through three different level play-throughs of the Puzzle Bloom game where four different camera systems are used. The first is using the old original camera systems, the second system is using the new camera system without the fuzzy decision maker, the third is using the new camera system with a crisp decision maker and the last system is using the new camera system with the fuzzy decision maker. First there will be a brief description of each camera system, and later the systems will be compared through five example scenarios.

6.1 Descriptions of the Camera Systems Used

Old Original Camera System

The original camera system uses a fixed top view camera and allows the player to rotate the camera around the up-axis and zoom in and out between a min. and max. distance from the avatar. This camera system is now referred to as camera system 1.

Third Person Follow-Behind Camera System

The new camera system is a third-person camera which follows the avatar from behind and smoothly moves between its current height, distance and orientation and the height, distance and orientation which is desired. This camera system is now referred to as camera system 2.

New Camera System With Crisp Decision Maker

This system is the same as camera system 2 but with a crisp decision maker. When

the crisp decision maker is enabled, the camera system tries to give camera hints to the player by including currently important objects which the avatar is nearby using the rules described in appendix D. This camera system is now referred to as camera system 3.

New Camera System With Fuzzy Decision Maker

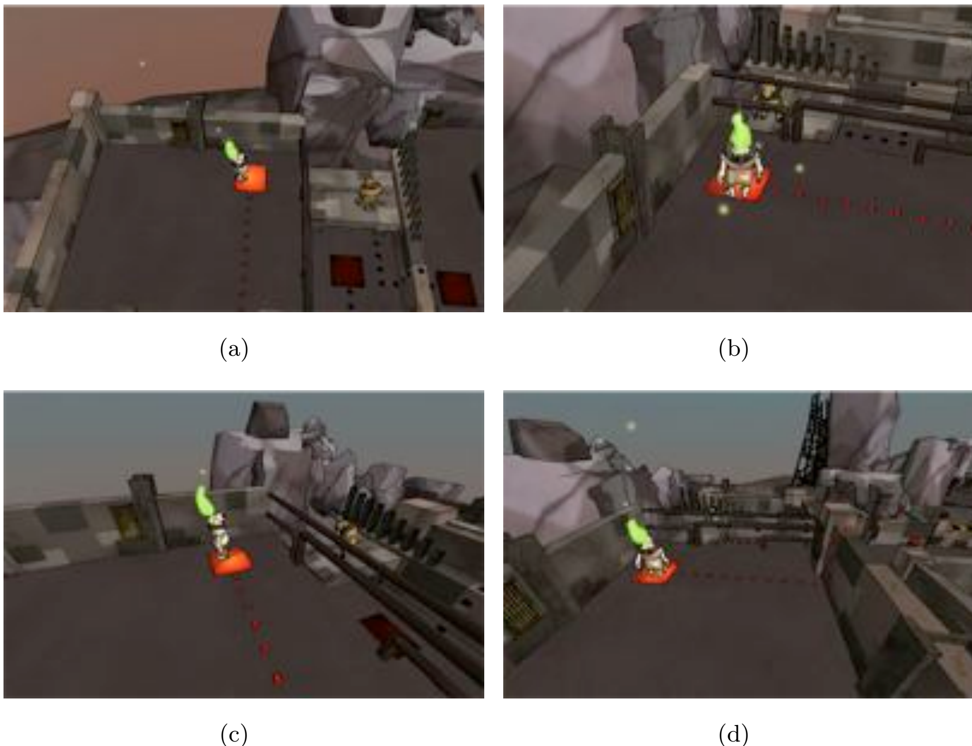
This system is the same as camera system 2 but with a fuzzy decision maker. When the fuzzy decision maker is enabled, the camera system tries to give camera hints to the player by including currently important objects which the avatar is nearby using the fuzzy system described in section 4.1. This camera system is now referred to as camera system 4.

6.2 Comparisons

This section presents a comparison of the four different camera systems by looking at each camera system in five example scenarios from the two playable levels in Puzzle Bloom and discussing which of the camera systems works best in each of the scenarios.

6.2.1 Example Scenario One

In the first scenario the player has just pressed a button which has opened a door but the player is not able to see what has actually happened. For unexperienced players, this might seem a bit confusing as there is no indication to what has just happened. This is one of the reasons why the line of red lights have been added to the game, to guide the player towards the door, but some players actually still miss this hint. Camera system 1 (figure 6.1(a)), 2 (figure 6.1(b)) and 3 (figure 6.1(c)) all fail to show the door which has been opened by pressing the button but camera system 3 (figure 6.1(d)) adjusts itself to include the opened door in the camera shot. In camera system 1, the player has to rotate the camera to see the door which not everybody does.

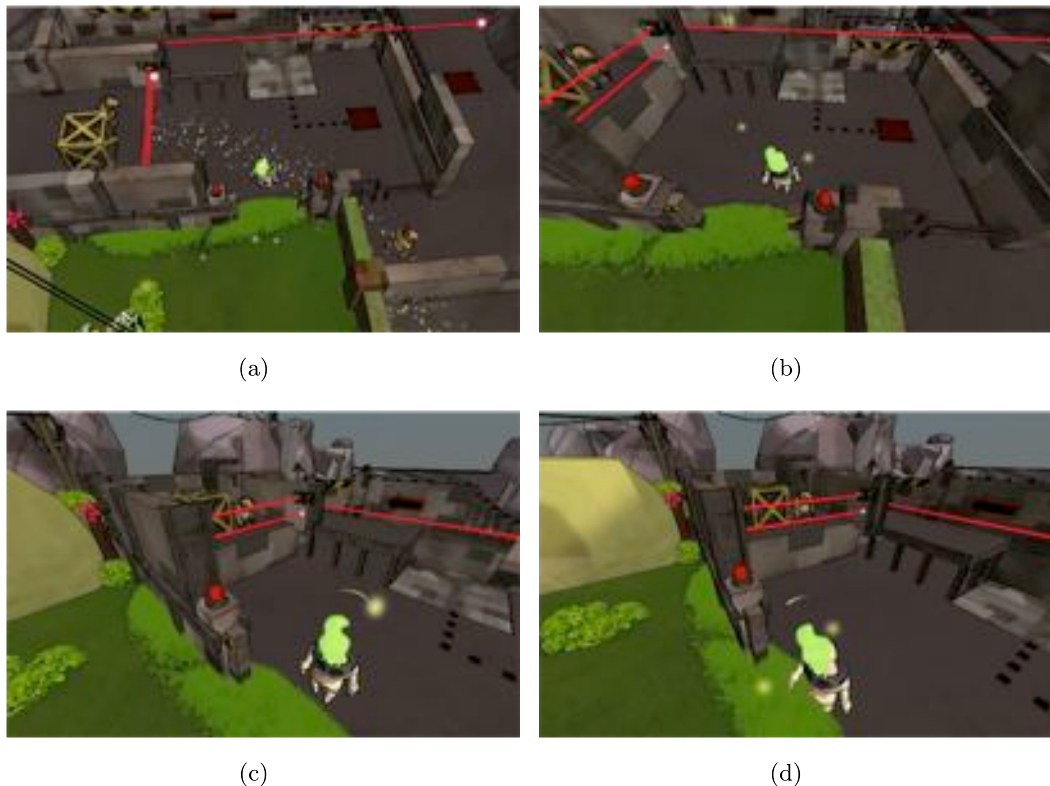


(a) (b) (c) (d)

Figure 6.1: Example scenario 1

6.2.2 Example Scenario Two

In scenario two, the player has entered a room with a quite complex puzzle and through the user tests of Puzzle Bloom, many players got stuck for quite some time before they figured it out. The first thing the player needs to do, is to push the crate down from the upper platform with the two lasers but many players missed this detail for some reason. In this situation, a camera hint would be very appropriate. Camera system 1 (figure 6.2(a)) gives a good overview of the room but because there is a lot of other objects in the room, there is a chance that these objects will disturb the player's attention. Camera system 2 (figure 6.2(b)) is closer to the avatar and therefore does not give the same overview as camera system 1 did. The screenshot shows the crate is only partly in the camera shot, which leaves the possibility of the player not noticing the crate. Both camera system 3 (figure 6.2(c)) and 4 (figure 6.2(d)) are closely zoomed in with the avatar in the foreground and the camera has turned slightly to put emphasis on the crate, but camera system 4 is more accurate in showing more of the crate in the camera shot.

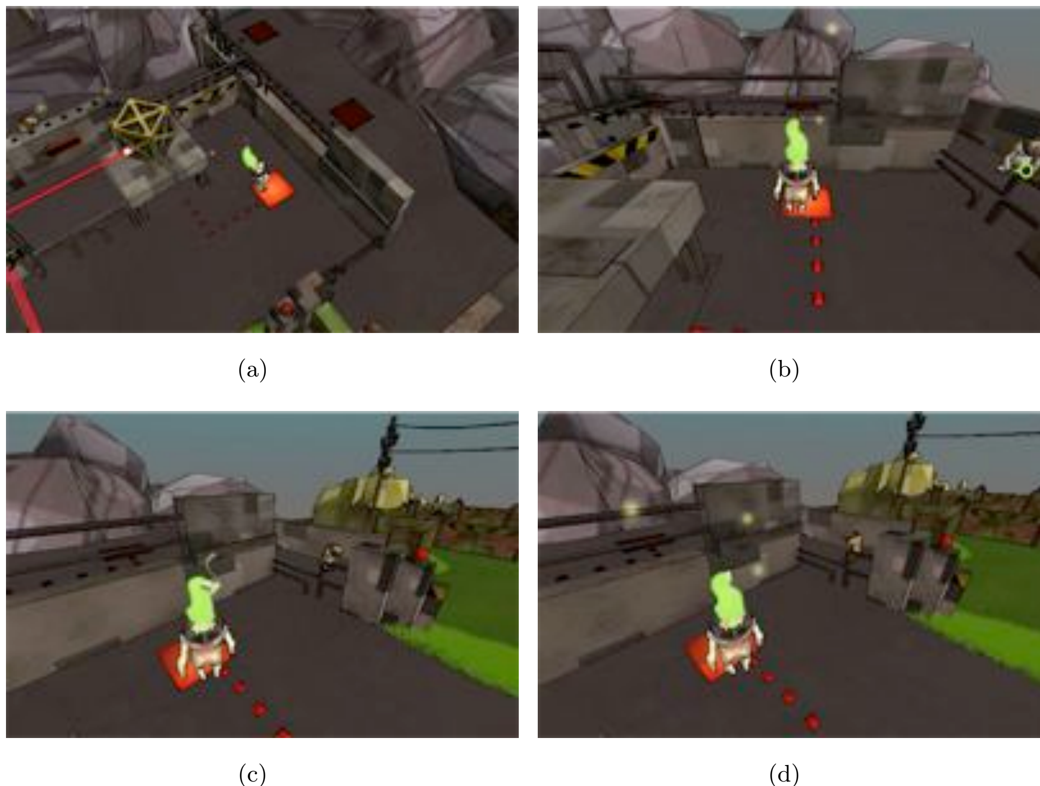


(a) (b) (c) (d)

Figure 6.2: Example scenario 2

6.2.3 Example Scenario Three

Another example where players were confused on what to do next is in scenario three. Again the players had missed the detail about the creature to the right of the avatar otherwise they might have tried to go in that direction which is first required later in the game. Camera system 1 (figure 6.3(a)) has a good overview of the room but the emphasis on the now relevant creature could be better as it is close to being cut away from the shot. Camera system 2 (figure 6.3(b)) has no overview and the important creature is again almost cut out from the camera shot. Camera system 3 (figure 6.3(c)) and 4 (figure 6.3(d)) again show the ability to turn the camera and put more focus on the relevant object.



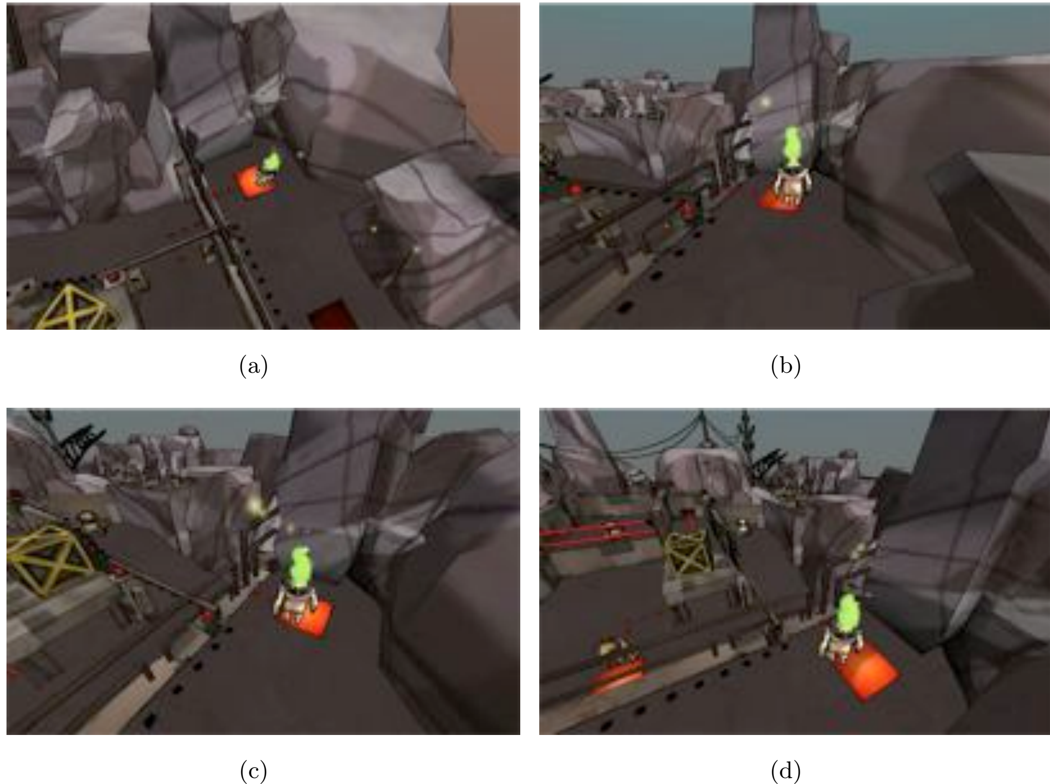
(a) (b) (c) (d)

Figure 6.3: Example scenario 3

6.2.4 Example Scenario Four

This scenario is another good example on how the old camera system manages to show a really important object without letting the player rotate the camera. The next step

the player should do after stepping on the button, is to switch to a creature which is not visible at all which forces the player to make the rotation in camera system 1 (figure 6.4(a)). In camera system 2 (figure 6.4(b)) the player can just turn to see the important creature but might not do that thinking it is a dead end. This is where camera system 3 (figure 6.4(c)) and 4 (figure 6.4(d)) helps the player by turning the camera automatically to show the creature the player needs to switch to.

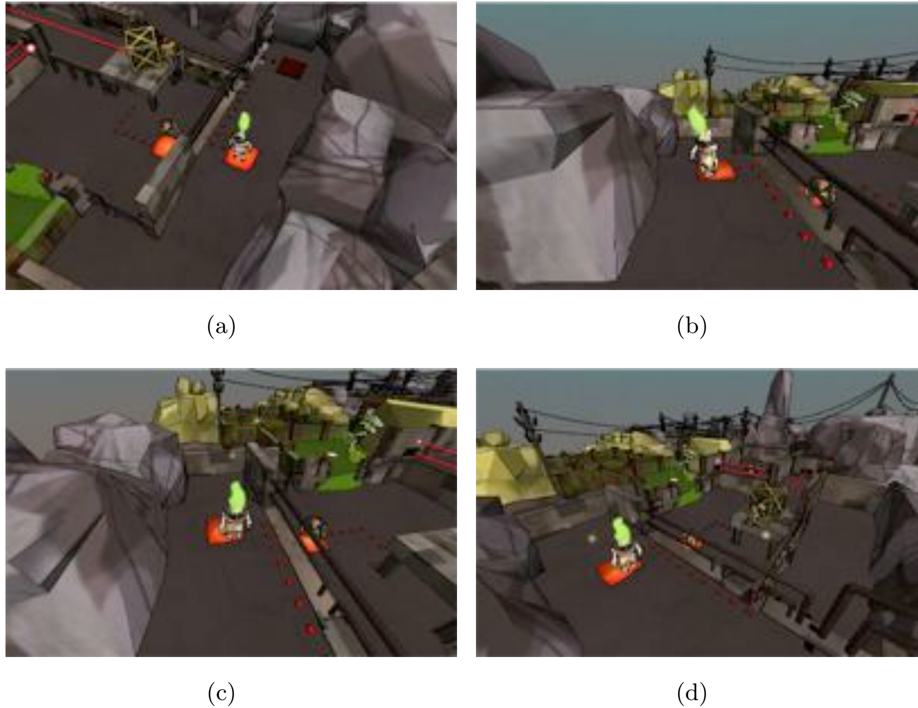


(a) (b) (c) (d)

Figure 6.4: Example scenario 4

6.2.5 Example Scenario Five

The last example again forces the player to rotate the camera when using the camera system 1 (figure 6.5(a)), as the opening door is hardly visible in the camera shot. The red lights should help to guide the player but not all players notice them. Many players during the user tests of Puzzle Blooom were not sure what to do when they pressed this button in this scenario. In camera system 2 (figure 6.5(b)) and 3 (figure 6.5(c)) it is even harder to see what is going on and it really needs the overview and guidance which camera system 4 (figure 6.5(d)) gives.



(a) (b) (c) (d)

Figure 6.5: Example scenario 5

6.3 Summary

As seen in all five example scenarios, camera system 4 (the one developed during this thesis) succeeded in showing the important nearby objects to the player and thereby helping and guiding the player throughout the game, where as either both or at least one of camera system 1 and 2 failed to do this. Camera system 4 could have used other camera shots to show the important objects but the shots chosen respect the player's intentional view. This makes the camera hints less annoying if the player is not so interested in using the hints but rather wants to explore the game world, which is one of the requirements for the developed camera system for this thesis.

Furthermore, the comparison between camera system 3 (with the crisp decision maker) and 4 shows that a fuzzy decision maker handles more complex situations better than a crisp decision maker as camera system 3 proves not to be able to find an acceptable view for the player in example scenario 1 and 5. The fuzzy decision maker can make more precise decisions than the crisp one (for example in scenario 2). Though it cannot really be seen from the example screenshots when playing with the crisp decision maker, the thresholds between the crisp sets were sometimes very obvious. For more examples see appendix C.

Chapter 7

Discussions

This chapter presents a critical assessment of this work, discusses open problems and proposes fields of future research.

7.1 Critical Assessment

The work on this thesis consists of three main parts. The first goal is to research and analyse existing camera systems and existing AI techniques in order to find an appropriate camera system and AI technique to base the new camera system and the decision maker on. Though it was possible to find works camera system from the research field, it is still very limited as to how much material you can find and it would have been interesting to have found works not just from the research area but works which have been applied to actual commercial games. The released works from the industry is almost nonexistent. The chosen AI technique could have been more advanced than it is or could have been combined with other techniques which will be discussed later in the future works section. Though the technique is simple, it still seems to choose some good camera views for the player in real-time.

The second goal of this thesis is to create a working implementation of the camera system with the decision maker. Though the implemented prototype already shows some satisfying and interesting results, a lot of areas in the camera system are still very basic for solving the involved problems. For example, the constraint solver only chooses the first solution which satisfies all the constraints but there might be a better solution in the search space which has not been found yet.

The last goal was to evaluate the developed camera system and decision maker. This was done through five example scenarios in the game Puzzle Bloom and by comparing the old camera system, the new system without the decision maker and the new system

with the decision maker (crisp and fuzzy) and by seeing how the camera views were created by the different systems. Even though this method can evaluate the developed camera system to some extent, it would have been better to use user tests to evaluate the system but unfortunately there was not enough time to perform the tests.

7.2 Future Work

This section presents possible directions for future research. First, it could be interesting to teach the decision maker when to use camera hints based on the player's behaviour. Some players might not want to receive hints while others may. This could for example be done by training a neural network to recognize how the player behaves. The neural network would then be able to tell if the player is exploring (and is not interested in the hints), if the player is lost (and could need some help) or if the player is really good at solving the puzzle (and does not need help).

As mentioned earlier, it would be good to perform tests to see if the users prefer the camera system with the decision maker as opposed to the other systems and if it actually helps them progress in the game or not. Questionnaires could be created to ask the users what system they preferred and if there were any dislikes about the system. To avoid the users giving biased answers, some users could start playing the game with the old system while some could start with the new system and then switch to playing a new level with the other system. It could also be tested to see how well users playing with the new system performed in terms of how quickly they figured out the puzzles compared with the old system. The user tests have not been performed as it takes too much time and it is outside the scope of this thesis.

The fuzzy rules and fuzzy sets used for the evaluation have been picked by the author using 'expert' knowledge and it has not been tested if these rules and sets are the optimal ones. Testing and evaluation should be done to find the best fuzzy sets and fuzzy rules.

Finally, the constraint solver could also be developed further. At the moment it is only using the first solution it finds which might not always be the best one. Some cinematographic rules could be used to evaluate each solution's cinematographic fitness. It might also be necessary to test for partial visibility of the scene objects and the avatar, as the camera system at the moment only checks for the visibility of the center of the objects, and therefore it might rule out solutions that could potentially be useful.

7.3 Summary

In this thesis, a camera system for computer games has been developed as well as a decision maker based on a fuzzy system capable of choosing camera views for the player which includes important game objects while still respecting the player's intentional view. Recent works on camera systems have been researched to allow the author to choose a suitable type of camera system to base the new one on. AI techniques have been analyzed to allow the author to pick an appropriate technique to base the decision method on. Based on these findings, the basic structure and algorithm for the camera system have been presented as well as detailed information regarding how the decision method works including details of the setup of the back-lying fuzzy system. To provide an example of the practicality of the developed system, the camera system has been integrated into the computer game Puzzle Bloom, in which the new system has been compared with both the old system and the new system without the decision maker through five different example scenarios. It has also been proven through the scenarios that a fuzzy decision maker performs better than a crisp decision maker. To conclude, the work of this thesis has been discussed and possible fields for future work have been presented.

Appendix A

Fuzzy System Setup for FLCam

This section contains all the fuzzy rules and FLVs used for the developed camera system.

A.1 Fuzzy Rules Used in FLCam

Fuzzy rules used for finding a crisp shot weight value

1. IF *Angle_VerySmall* AND *Dist_Close* THEN *SW_Low*
2. IF *Angle_VerySmall* AND *Dist_Medium* THEN *SW_Medium*
3. IF *Angle_VerySmall* AND *Dist_Far* THEN *SW_High*
4. IF *Angle_Small* AND *Dist_Close* THEN *SW_Low*
5. IF *Angle_Small* AND *Dist_Medium* THEN *SW_Medium*
6. IF *Angle_Small* AND *Dist_Far* THEN *SW_High*
7. IF *Angle_Medium* AND *Dist_Close* THEN *SW_Medium*
8. IF *Angle_Medium* AND *Dist_Medium* THEN *SW_Medium*
9. IF *Angle_Medium* AND *Dist_Far* THEN *SW_High*
10. IF *Angle_Large* AND *Dist_Close* THEN *SW_Medium*
11. IF *Angle_Large* AND *Dist_Medium* THEN *SW_High*
12. IF *Angle_Large* AND *Dist_Far* THEN *SW_High*

13. IF $Angle_VeryLarge$ AND $Dist_Close$ THEN SW_Medium
14. IF $Angle_VeryLarge$ AND $Dist_Medium$ THEN SW_High
15. IF $Angle_Large$ AND $Dist_Far$ THEN SW_High
16. IF $Contrib_Low$ AND THEN SW_High
17. IF $Contrib_Medium$ THEN SW_Medium
18. IF $Contrib_High$ THEN SW_Low

Fuzzy rules used for finding a crisp frame coherence value

1. IF $DiffAngle_VerySmall$ AND $DiffDist_Close$ THEN FC_Low
2. IF $DiffAngle_VerySmall$ AND $DiffDist_Medium$ THEN FC_Medium
3. IF $DiffAngle_VerySmall$ AND $DiffDist_Far$ THEN FC_Meidum
4. IF $DiffAngle_Small$ AND $DiffDist_Close$ THEN SW_Low
5. IF $DiffAngle_Small$ AND $DiffDist_Medium$ THEN SW_Medium
6. IF $DiffAngle_Small$ AND $DiffDist_Far$ THEN SW_Medium
7. IF $DiffAngle_Medium$ AND $DiffDist_Close$ THEN SW_Low
8. IF $DiffAngle_Medium$ AND $DiffDist_Medium$ THEN SW_Medium
9. IF $DiffAngle_Medium$ AND $DiffDist_Far$ THEN SW_High
10. IF $DiffAngle_Large$ AND $DiffDist_Close$ THEN SW_Medium
11. IF $DiffAngle_Large$ AND $DiffDist_Medium$ THEN SW_Medium
12. IF $DiffAngle_Large$ AND $DiffDist_Far$ THEN SW_High
13. IF $DiffAngle_VeryLarge$ AND $DiffDist_Close$ THEN SW_Medium
14. IF $DiffAngle_VeryLarge$ AND $DiffDist_Medium$ THEN SW_High
15. IF $DiffAngle_Large$ AND $DiffDist_Far$ THEN SW_High

Fuzzy rules used for finding a crisp distance weight value

1. IF *Dist_Close* THEN *DW_High*
2. IF *Dist_Medium* THEN *DW_Medium*
3. IF *Dist_Far* THEN *DW_Low*

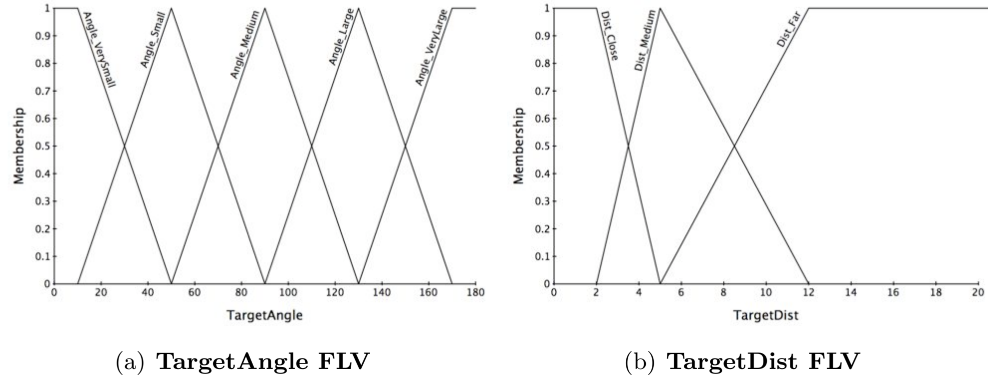
Fuzzy rules used for finding a crisp height weight value

1. IF *Dist_Close* THEN *HW_High*
2. IF *Dist_Medium* THEN *HW_Medium*
3. IF *Dist_Far* THEN *HW_Low*

Fuzzy rules used for finding a crisp orientation weight value

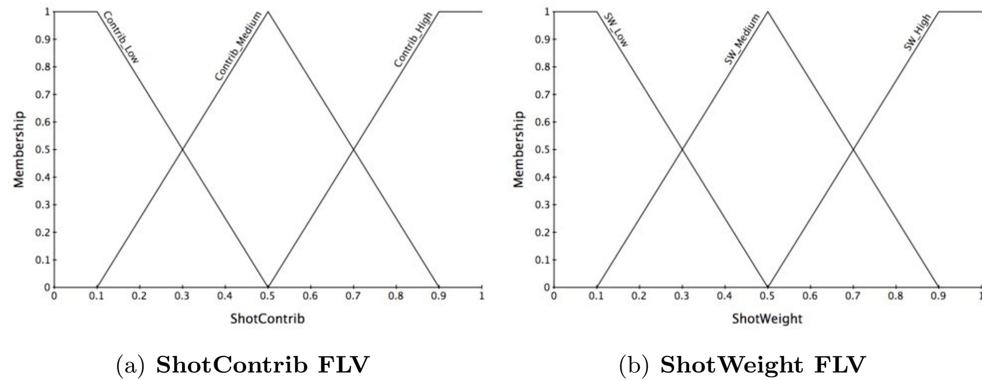
1. IF *Dist_Close* THEN *OW_High*
2. IF *Dist_Medium* THEN *OW_Medium*
3. IF *Dist_Far* THEN *OW_Low*

A.2 FLVs Used in FLCam



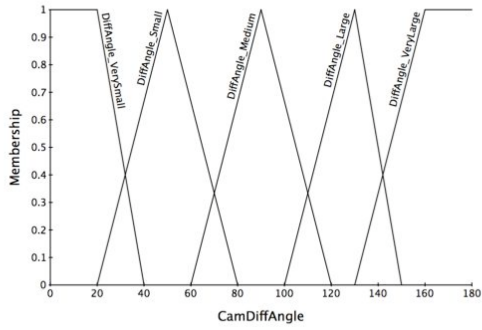
(a) **TargetAngle FLV** (b) **TargetDist FLV**

Figure A.1: (a) is the angle between the direction vector from the avatar's position to the look-at point and the avatar's forward vector and (b) is the distance between the look-at point the avatar's predicted future position

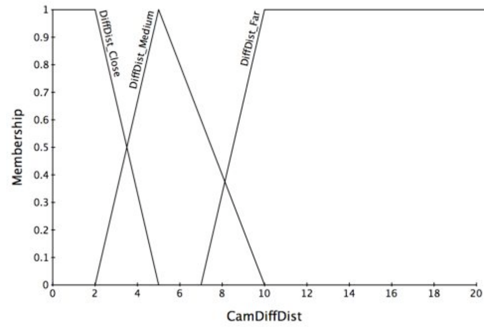


(a) **ShotContrib FLV** (b) **ShotWeight FLV**

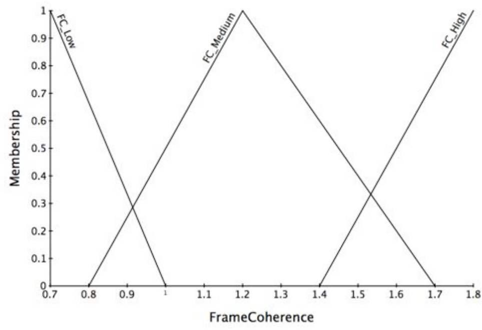
Figure A.2: (a) is the average shot contribution of all the SOs in the current shot and (b) is the interpolation value between the optimal view and the player's intentional view, where 0 is completely the optimal view and 1 is completely the player's view



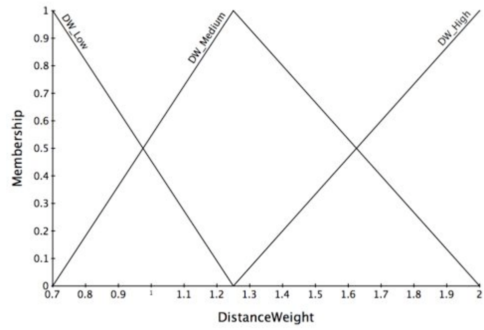
(a) **CamDiffAngle FLV**



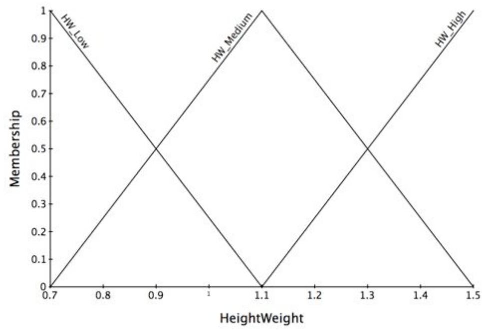
(b) **CamDiffDist FLV**



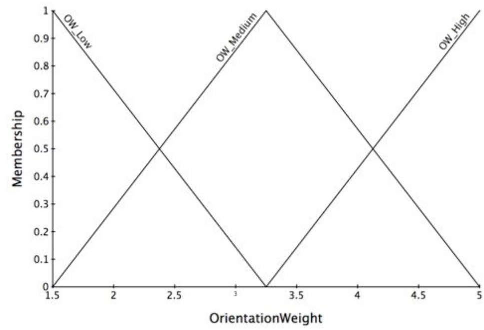
(c) **FrameCoherence FLV**



(d) **DistanceWeight FLV**



(e) **HeightWeight FLV**



(f) **OrientationWeight FLV**

Figure A.3: (a) is the angle between the camera's current forward vector and the camera's ideal forward vector, (b) is the distance between the camera's current position and the camera's ideal position, (c) is the frame coherence weight for the camera, (d) is the distance weight for the camera, (e) is the height weight for the camera and (f) is the orientation weight for the camera

Appendix B

Object & Shot Rules

These are the object and shot rules used for the project. Other rules could easily be added or existing ones could be modified.

```
        <XML version="1.0">

<RULE name="Establishing Shot" appliesTo="objects">

<CONDITION>
    List<ISceneObject> relevantObjects = cs.SceneObjects;

    if(relevantObjects.Count == 0)
        return false;

    bool isNewScene = true;
    foreach(ISceneObject so in relevantObjects)
    {
        if(cs.CountNumberOfTimesOnScreen(so) > 0)
        {
                isNewScene = false;
                break;
        }
    }

    return isNewScene;

</CONDITION>

<ACTION>
    List<ISceneObject> relevantObjects = cs.SceneObjects;

    IShot newShot = cs.NewShot(relevantObjects, 0.9f);

    cs.QueueShot(newShot);
```

```
</ACTION>
</RULE>

<RULE name="New Event Shot" appliesTo="objects">

<CONDITION>

    List<ISceneObject> relevantObjects = cs.SceneObjects;

    return relevantObjects.Count > 0;

</CONDITION>

<ACTION>

    bool addShot = true;
    List<IShot> shotHistory = cs.ShotHistory;

        foreach(IShot oldShot in shotHistory)
        {
                if(cs.CompareShots(oldShot, newShot) == 0)
                {
            if(cs.GetCharAngleToScene(newShot) > cs.IgnoreShotAngle)
            {
                addShot = false;
                break;
            }
                }
        }

    if(addShot)
        cs.QueueShot(newShot);

</ACTION>
</RULE>

<RULE name="Default Shot" appliesTo="objects">

<CONDITION>

    return true;

</CONDITION>

<ACTION>

    IShot newShot = cs.NewShot(new List<ISceneObject>(), 0.0f);

    cs.QueueShot(newShot);

</ACTION>
```

```
</RULE>

<RULE name="Object Visibility" appliesTo="shots">

<CONDITION>

    cs.ObjectsVisible = false;

    IShot currentShot = cs.CurrentShot;

    bool allVisible = true;

    foreach(ISceneObject so in currentShot.SceneObjects)
    {
        RaycastHit hit;

        if(so.MustBeVisible && Physics.Linecast(cs.Position,
                so.Position, out hit, cs.Mask.value))
        {
                allVisible = hit.transform.IsChildOf(so.GetTransform)
                        || hit.transform == so.GetTransform;
        }
        else
        {
                allVisible = true;
        }

        if(!allVisible)
                break;
    }

    return allVisible;

</CONDITION>

<ACTION>

    cs.ObjectsVisible = true;

</ACTION>
</RULE>

<RULE name="Character Visibility" appliesTo="shots">

<CONDITION>

    cs.TargetVisible = false;

    RaycastHit hit;

    if(Physics.Linecast(cs.Position, cs.TargetPosition,
        out hit, cs.MaskB.value))
```

```
    {
        if(!hit.transform.IsChildOf(cs.Target.GetTransform)
                || hit.transform != cs.Target.GetTransform)
        {
                return false;
        }
    }

    return true;

</CONDITION>

<ACTION>

    cs.TargetVisible = true;

</ACTION>
</RULE>

<RULE name="OneEighty Degree Line" appliesTo="shots">

<CONDITION>

    if(!cs.ObjectsVisible || !cs.TargetVisible)
    {
        float r = 0;
        float step = 180.0f / (cs.MaxIterations / 2.0f);

        float c = cs.IterationCount % 2 == 0 ? -1 : 1;
        cs.Rotation = cs.SaveCamRotation.rotation;
        cs.Position = cs.SaveCamRotation.position;
        float cc = cs.IterationCount*0.5f*step;
        if(cs.IterationCount % 2 != 0)
        {
                int i = cs.IterationCount / 2;
                cc = cs.IterationCount*step - i*step;
        }
        r = c * cc;

        cs.SetupSmoothFollow(cs.IterationCamAngle, cs.Radius, r, true);

        return false;
    }
    else
    {
        return true;
    }
</CONDITION>
<ACTION>
</ACTION>
</RULE>
</XML>
```

Appendix C

Puzzle Bloom Screenshots

The following images are screenshots taken from the game Puzzle Bloom with the use of the original camera system, the new developed camera system without a decision maker, the new system with a crisp decision maker and the new system with a fuzzy decision maker. These are more examples of the differences between the four camera systems.

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

Figure C.1: Extra scenario 1 ((a) - (d)), the first puzzle of the game where the player learns how to switch to another creature. Extra scenario 2 ((e) - (h)), demonstrates how well the camera systems adapt to when important objects are almost behind the player.

Figure C.2: Extra scenario 3 ((a) - (d)) has both the door and the other creature as important objects because the player needs to switch to the other creature to get through the door. Extra scenario 4 ((e) - (h)), shows the newly opened door to the next part of the game as an important object.

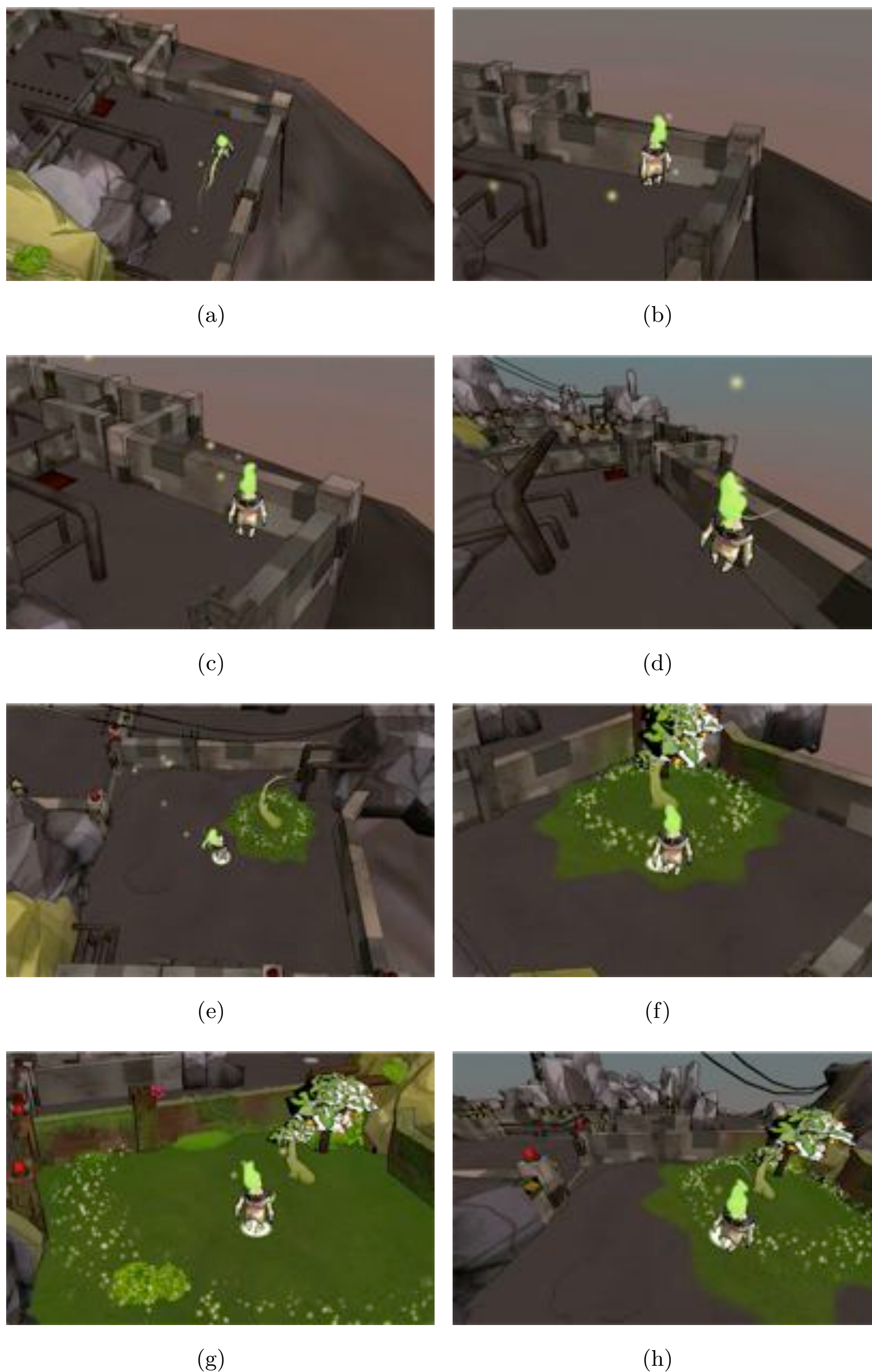


Figure C.3: Extra scenario 5 ((a) - (d)) demonstrates how well the camera systems adapt to when important objects are almost perpendicular to the player's viewing direction. Extra scenario 6 ((e) - (h)) demonstrates how it is only the camera system with the fuzzy decision maker, which gives a perfect view of the important object in the scene when there is a large angle and distance to the important object.

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

Figure C.4: Extra scenario 7 ((a) - (d)) shows the player's first experience with the crate (which is the important object) and the player is forced to learn how the crate can be used. Extra scenario 8 ((e) - (h)) shows how the fuzzy camera system gives the best view of the important creature which the player needs to switch to next.

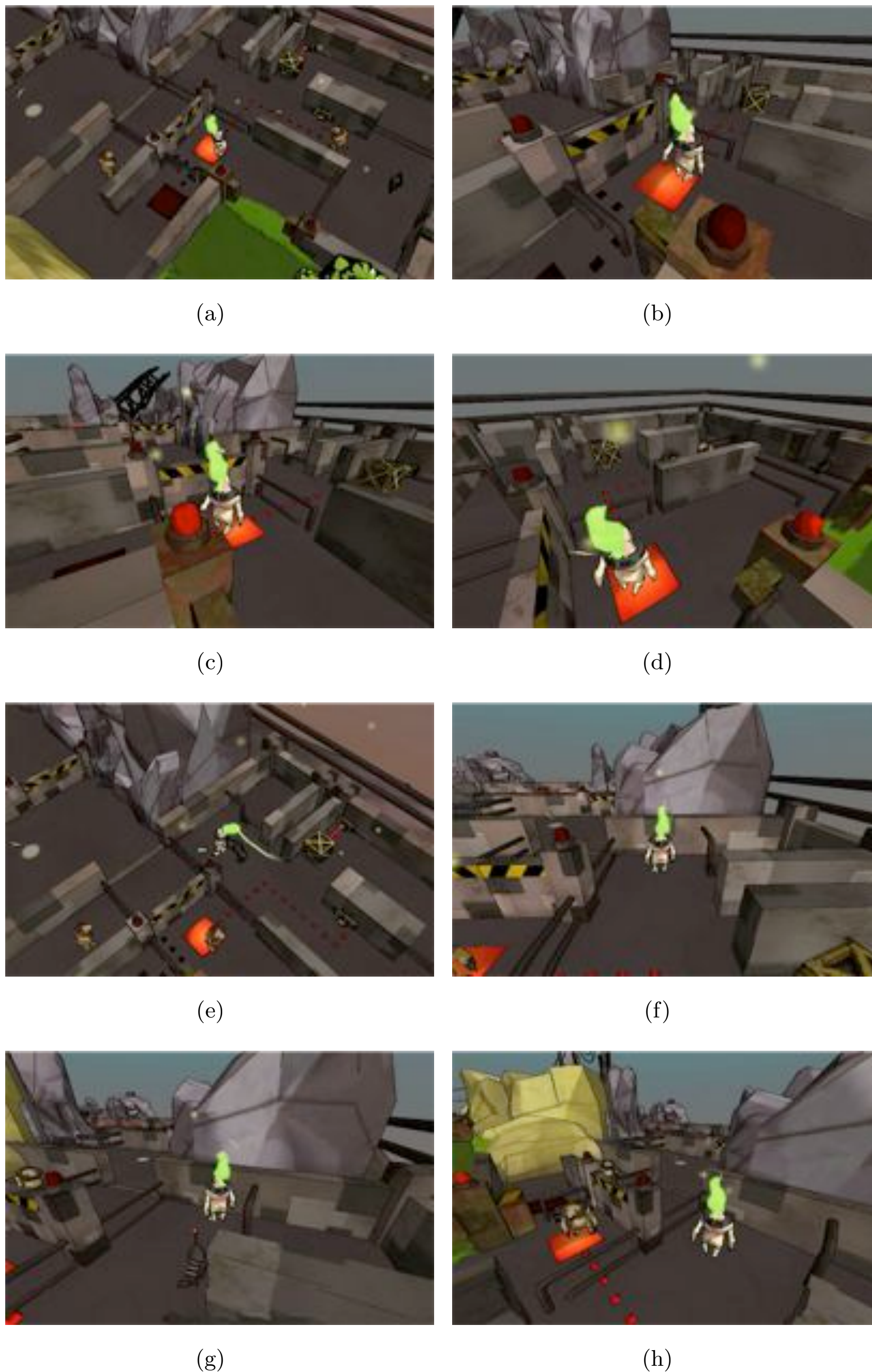
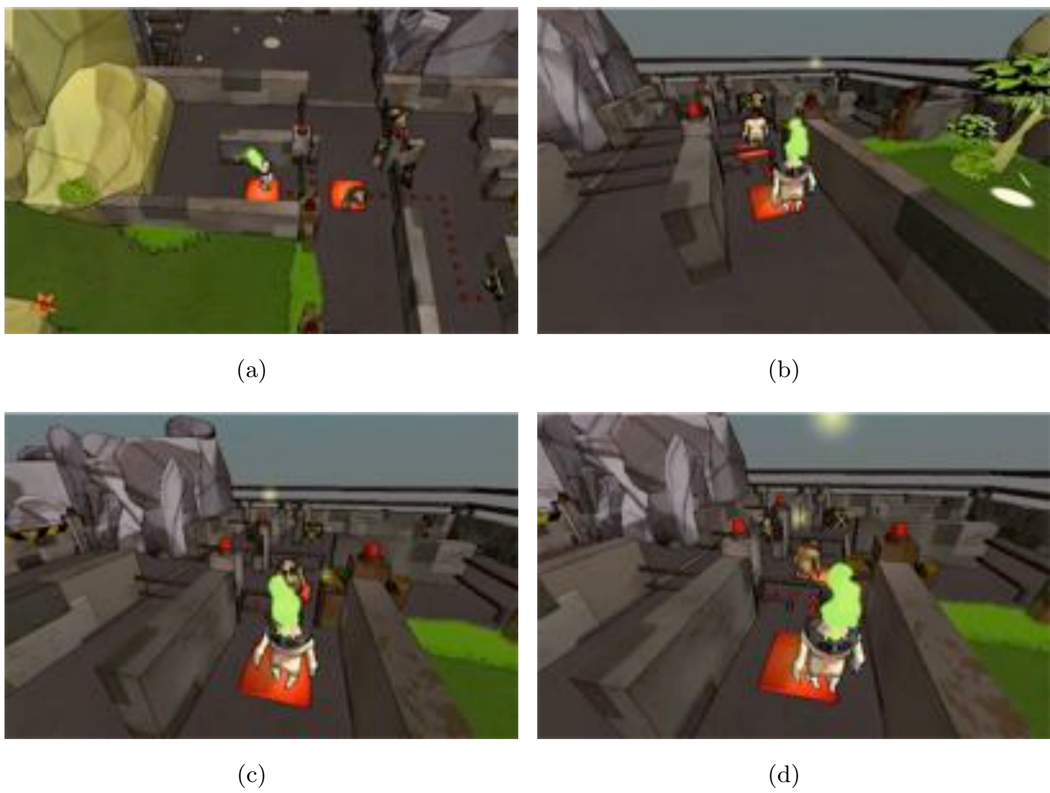


Figure C.5: Extra scenario 9 ((a) - (d)) shows how the fuzzy camera system gives the best view of the creature which becomes important once the button is pressed. Extra scenario 10 ((e) - (h)) shows how the fuzzy camera system gives the best view of the important creature which the player needs to switch to.



(a) (b) (c) (d)

Figure C.6: Extra scenario 11 ((a) - (d)) demonstrates how well the camera systems show the door which becomes important when the button is pressed.

Appendix D

Rules for the Crisp Decision Maker

This chapter describes the rules used for the crisp decision maker which is used in the comparison between itself and the fuzzy decision maker during the evaluation to show that it is better to use fuzzy rules rather than crisp rules for the decision maker. The rules (to find the shot weight) are evaluated in the following order:

1. **IF** *angle* >= 150 AND *distance* >= 8.5 **THEN** return 1;
2. **IF** *angle* >= 150 AND *distance* >= 3.5 **THEN** return 1;
3. **IF** *angle* >= 150 AND *distance* < 3.5 **THEN** return 0.7;
4. **IF** *angle* >= 110 AND *distance* >= 8.5 **THEN** return 1;
5. **IF** *angle* >= 110 AND *distance* >= 3.5 **THEN** return 0.8;
6. **IF** *angle* >= 110 AND *distance* < 3.5 **THEN** return 0.5;
7. **IF** *angle* >= 70 AND *distance* >= 8.5 **THEN** return 1;
8. **IF** *angle* >= 70 AND *distance* >= 3.5 **THEN** return 0.7;
9. **IF** *angle* >= 70 AND *distance* < 3.5 **THEN** return 0.3;
10. **IF** *angle* >= 30 AND *distance* >= 8.5 **THEN** return 0.8;
11. **IF** *angle* >= 30 AND *distance* >= 3.5 **THEN** return 0.3;
12. **IF** *angle* >= 30 AND *distance* < 3.5 **THEN** return 1;

13. **IF** *angle* < 30 AND *distance* >= 8.5 **THEN** return 0.8;
14. **IF** *angle* < 30 AND *distance* >= 3.5 **THEN** return 0.3;
15. **IF** *angle* < 30 AND *distance* < 3.5 **THEN** return 0;

Appendix E

Summary of the Project Process

The final problem formulation and method has changed to what is stated in the approved project agreement in the project base.

After handing in the project agreement, the author did not know what kind of game he would end up with in his DADIU production in March 2009. After finishing the DADIU game the author realized that the game was well suited to use a test bed for what he wanted to do. After doing more research on the area of camera control, the author decided to concentrate on something that he had described in the project agreement. The work by Cozic [1] inspired the author on the change of the central problem area.

The final problem formulation is now: How can a method be developed using an AI technique to make decisions on what camera view to use by both respecting the camera system's optimal view and the player's own intentional view?

The method, which ended up being used was as follows:

1. **Research:**
   (a) Investigating previous related works and methods using the research field and the industry
   (b) Investigating relevant AI theories, which could be used as the intelligent decision maker
2. **Design:**
   (a) A camera system will designed based on the most appropriate system found from the investigated works and methods
   (b) An intelligent decision making method will be designed based on the most appropriate AI theory found from the investigated AI theories

3. **Implementation:**
    (a) The designed system and method will be implemented in the Puzzle Bloom game made during the DADIU March 2009 production
4. **Evaluate:**
    (a) The following three example game sessions (using the Puzzle Bloom game) will be used to evaluate the outcome of the thesis:
        i. A level play-through using the old camera system
        ii. A level play-through using the new camera system without the intelligent decision maker
        iii. A level play-through using the new camera system with the intelligent decision maker
5. **Conclusion and Discussion:**
    (a) Based on the evaluation results in comparison to the aim of the thesis, the success of the thesis will be discussed

Glossary

180° Line or **Line of Action**
An imaginary line used to help stage camera positions for shooting action. Typically 'drawn' along the line of sight between two characters in a scene, or following the movement of characters, cars etc.

Camera Angle
The angle at which a camera views a particular scene. Camera angle can be based on horizontal camera positioning around the subject or vertical positioning below or above the subject

Camera Setup
A place on the film set where a camera is positioned to record a shot. Each time the camera is physically moved to a new position it is considered a new camera setup

Dolly
When moving the camera in the horizontal plane

Field of View
The angular extent of the observable world that is seen at any given moment. There exist both a vertical and horizontal field of view

Pan
When pivoting the camera on its vertical axis

Tilt
When pivoting the camera on its horizontal axis

References

[1] Laurent Cozic. **Automated Cinematography for Games**. June 2007. iii, 12, 13, 16, 40, 41, 76

[2] Marc Christie and Patrick Olivier. **Camera Control in Computer Graphics**. 2006. 4

[3] A. Corrigan. **A simple third-person camera using the polar coordinate system**. November 2001. 5

[4] J. Stone. **Third-person camera navigation**. pages 303–314, Charles River Media, 2004. 5

[5] P. Carlisle. **An AI approach to creating an intelligent camera system**. pages 179–185, Charles River Media, 2004. 5

[6] C. Reynolds. **Steering behaviours for autonomous characters**. 1999. 5

[7] N. Bobick. **Rotating objects using quaternions**. July 1998. 5

[8] J. Giors. **The Full Spectrum Warrior camera system**. 2004. 6

[9] Steven M. Drucker and David Zeltzer. **CamDroid: A System for Implementing Intelligent Camera Control**. In *Symposium on Interactive 3D Graphics*, pages 139–144, 1995. 6

[10] Steven M. Drucker, Tinsley A. Galyean, and David Zeltzer. **CINEMA: A System for Procedural Camera Movements**, 1992. 6

[11] William H. Bares, Joël P. Grégoire, and James C. Lester. **Realtime constraint-based cinematography for complex interactive 3D worlds**. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 1101–1106, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. 7

[12] William H. Bares and James C. Lester. **Intelligent multi-shot visualization interfaces for dynamic 3D worlds**. In *IUI '99: Proceedings of the 4th international conference on Intelligent user interfaces*, pages 119–126, New York, NY, USA, 1999. ACM. 7, 10

[13] William H. Bares, Somying Thainimit, and Scott Mcdermott. **A model for constraint-based camera planning**. In *In Smart Graphics. Papers from the 2000 AAAI Spring Symposium*, **16**, pages 84–91, 2000. 8

[14] William Bares, Scott Mcdermott, Christina Boudreaux, and Somying Thainimit. **Virtual 3D camera composition from frame constraints**. In *ACM Multimedia*, pages 177–186, 2000. 8

[15] Li W. He, Michael F. Cohen, and David H. Salesin. **The virtual cinematographer: a paradigm for automatic real-time camera control and directing**. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 217–224, New York, NY, USA, 1996. ACM. 8, 18

[16] N Halper and P Olivier. **CAMPLAN: A Camera Planning Agent**. In *Smart Graphics 2000 AAAI Spring Symposium*, pages 92–100, 2000. 8, 19

[17] G Luger and B. Stubblefield. **Artificial Intelligence: Structures and Strategies for Complex Problem Solving, third edition**. 1998. 9

[18] Nicolas Halper, Ralf Helbing, and Thomas Strothotte. **Managing the Trade-Off Between Constraint Satisfaction and Frame Coherence**. 9, 11, 12

[19] Owen Bourne and Abdul Sattar. **Applying Constraint Weighting to Autonomous Camera Control**, June 2005. 11

[20] Owen Bourne, Abdul Sattar, and Scott Goodwin. **A Constraint-Based Autonomous 3D Camera System**. *Constraints*, **13**(1):180–205, June 2008. 11, 13, 16, 41

[21] Er Hornung, Gerhard Lakemeyer, and Georg Trogemann. **Autonomous real-time camera agents in interactive narratives and games**. In *Proceedings of the IVA 2003: 4th International Working Conference on Intelligent Virtual Agents, 15.-17.9.2003, Irsee, Germany, Lecture Notes in Computer Science 2792*, **2792**, pages 236–243, 2003. 12, 20

[22] Brian Hawkins. **Creating an Event-Driven Cinematic Camera**. Gamasutra Website, 2003. 12, 41

[23] Alex J. Champandard. **AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors**. New Riders Publishing, 1249 Eighth Street, Berkeley, CA, 2003. 17

[24] David M. Bourg and Glenn Seeman. **AI for Game Developers**. OReilly, OReilly, 2004. 20

[25] Mat Buckland. **Programming Game AI by Example**. Wordware Publishing, Wordware Publishing, 2005. 21

[26] DADIU Team 4. **Puzzle Bloom**. The National Academy of Digital Interactive Entertainment, March 2009. 36

[27] William E. Combs. **The Combs Method For Rapid Inference**. The Boeing Company, 1997. 28

[28] Philip Nowell. **Player Position Prediction**. mathproofs.blogspot.com, 2005. 44