Unnatural Gaming

Checkmate

Project Definition

SEG 4000
Dr. Liam Peyton

Revision #: 03
Revision Date: January 29, 2003

Ashraf Ismail
Jacques Lebrun
Bretton MacLean
Rahwa Keleta

Table of Contents

1. Title

Team Name: Unnatural Gaming (UG)
Project Name: Checkmate

2. Team Members

Ashraf Ismail
Jacques Lebrun
Bretton MacLean
Rahwa Keleta

3. Customer

Our product is targeted for the market of online gamers. So we need a customer who belongs to this community. We have asked Adam Murray to fill this role.

Adam Murray:

Adam is a graduate student of the SITE department. He has been our TA for SEG 3120 and SEG 4100. He is also an avid gamer who suites our target customer group. This is why he has been chosen him to represent them.

4. System Description

We will be making a modification of the Unreal Tournament 2003® (UT2003) engine. The modification will use the Unreal engine[1] as a foundation. UT2003 provides a framework for third-party developers to build custom games. This is fully supported and endorsed by Epic Games[2], creators of UT2003.

The game we are developing is titled Checkmate. This will be an online team deathmatch game incorporating elements of Chess. Two teams will be pitted against each other for the sole purpose of eliminating the opposition's King. Users will have access to player classes based on chess pieces. Each team will have one king and any number of queens, rooks, knights, bishops and pawns. Unlike chess, which is turn-based, Checkmate will be a fast-paced, real-time FPS.

NOTE: The information below is new content that we are adding to UT2003.

Player Classes

As mentioned, the player classes are based on chess pieces. There are six in total: Pawn, Knight, Bishop, Rook, Queen and King. Each class has its own properties, strengths and weaknesses. A good portion of our testing will be based on balancing these classes, so that no class is too weak or too powerful relative to the other classes. For each class, we will be designing our own weapons and abilities. We will be using player models provided by the game for this project. *Table-1* shows a very brief breakdown of the player classes. *Note: we are still in the process of defining properties and skills of these classes.*

Player Class	Point Value	Description
Pawn	1	-Weakest of all the classes -Will be rewarded with the most amount of points
Knight	3	-Acrobatic skills

[1] More information about the Unreal engine can be found in *Appendix B*.
[2] Epic Games, www.epicgames.com .

		-Relatively short ranged weapon
Bishop	3	-The sniper -Fast -low amount of health
Rook	5	-Very tough, very strong -Slow
Queen	9	-Most powerful of all the classes -Can only obtain points by fragging another queen or king.
King	10	-The backbone of the team -A melee fighter -Amount of health scales with the amount of players on the opposing team (amount yet to be determined)

Table 1: Player Class Information

Point System

We have developed a point system that rewards players and advances the game. This system awards the player by giving them points based on their current player class and the player class of the user they have fragged. This point system uses the point value of chess pieces as the basis for its calculations. See *Table-1* for the point values, these are the actual point values used in Chess.

The logic for the calculation is:

if (EnemyClass ≥ PlayerClass)

$$\text{PointsAwarded} = ceiling\left[\frac{EnemyClass}{PlayerClass}\right]$$

else

PointsAwarded = 0

Equation 1: Point System Formula

This will help balance out the playing field while at the same time rewarding the skilled players. If someone earns enough points to become the queen (who is very powerful) they will be rewarded with very little amount of points. Yet those who play the pawn (weakest of all the player classes) will be rewarded with the most points.

As the game progresses, players will have accumulated more points and thus be able to use more powerful player classes.

Player Class Selection

The same point values are used for selecting a player class. For example: to select the queen, the user must have 9 points. The only cases where the points are different are the pawn (who is free for selection), and the king (who is automatically selected).

The actual selection is done either through a menu that the player brings up during gameplay or upon death where a menu will pop up (the two menus are the same). Binds can also be setup for this.

A match is ended when a king dies, and the player who deals the final blow will be awarded his point value. The player with the highest amount of accumulated points at the end of the round will be the king for the next round (for both teams). When a player becomes a king, their points are reset to zero, this allows all players the opportunity to play as the king.

Matches and Rounds

Matches will be based either on the time limit, score limit, a combination of the two, or a round limit. Each match will have multiple rounds. A round is played until one team's King dies. The victorious team is awarded a point for the team score and a new round begins. This continues until the end of a match where a wining team

is declared (the team who has eliminated the king the most). At this point a new match begins resetting the scores.

We are still in the process of setting up the rules for assigning a king in special circumstances. These scenarios are:

- User playing a king disconnects
- At the end of a round, multiple users are tied for first place in points

AI

UT2003 comes with computer-controlled opponents, called "bots". We will be enhancing these bots to function in the Checkmate game type (select class, behave differently depending on which class is used, and play according to the rules of the game). During the course of development, we will be using the bots to facilitate some testing.

Use Case Scenarios

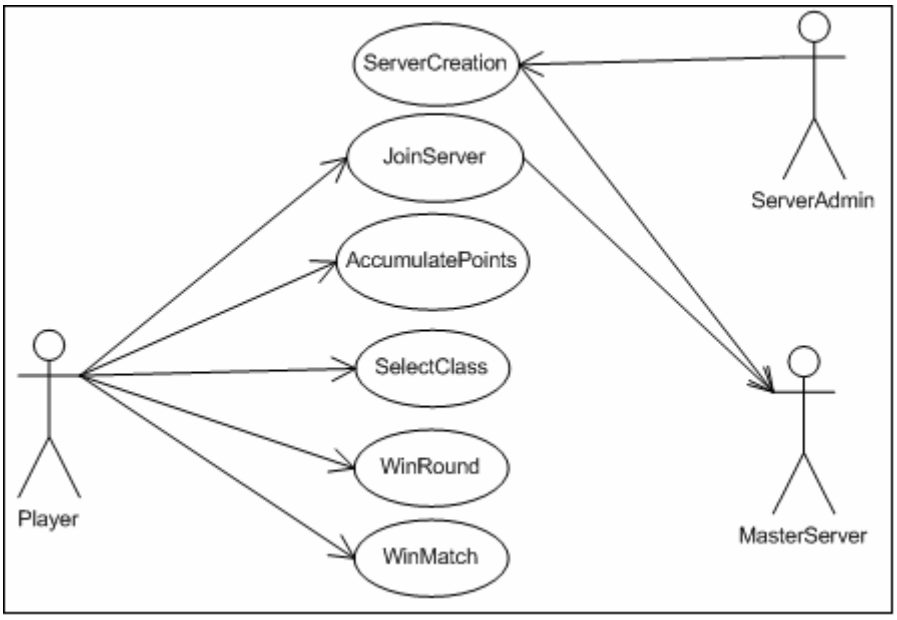


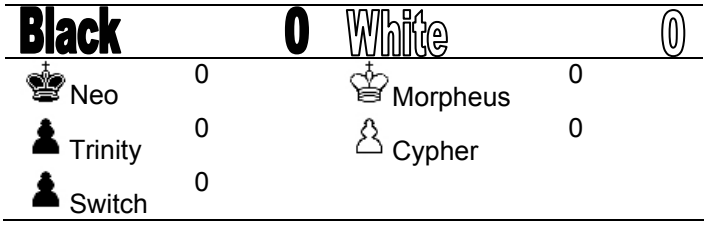
Figure 1: Use Case Diagram

These use cases are described by the example below.

1. Server creation
    1.1. Server *"Killa Checkmate Server"* starts up.
        1.1.1. The server administrator selects the initial map and starts the game server.
        1.1.2. The master server is notified of the new server and updates its server list.

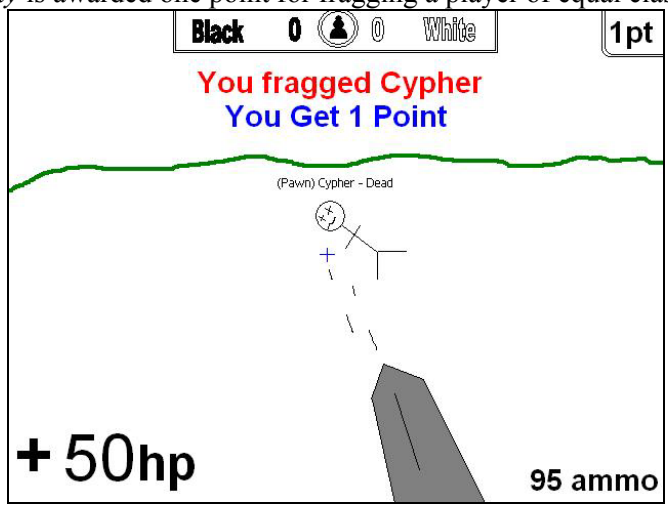
2. Players join the Checkmate server
    2.1. Player *"Neo"* joins the server.
        2.1.1. The list of available servers is obtained from the master server; *Neo* selects *"Killa Checkmate Server"* from the list of servers.
        2.1.2. The master server authenticates the client and allows the connection.
        2.1.3. *Neo* selects a team to join, he chooses the Black team.
        2.1.4. He is assigned a starting player class.
            2.1.4.1. Since the Black team is currently empty, he spawns in as the Black King.
    2.2. Player *"Morpheus"* joins the server.
        2.2.1. Repeat 2.1.1. & 2.1.2.
        2.2.2. *Morpheus* selects the White team.

2.2.3. He is assigned a starting player class.
2.2.3.1. Since the White team is currently empty, he spawns in as the White King.
2.3. Player *"Trinity"* joins the server.
2.3.1. Repeat 2.1.1. & 2.1.2.
2.3.2. *Trinity* selects the Black team.
2.3.3. She is assigned a starting player class.
2.3.3.1. Since the Black team already has a King, she spawns in as a Pawn.
2.4. Player *"Cypher"* joins the server.
2.4.1. Repeat 2.1.1. & 2.1.2.
2.4.2. *Cypher* selects the White team.
2.4.3. He is assigned a starting player class.
2.4.3.1. Since the White team already has a King, he spawns in as a Pawn.
2.5. Player *"Switch"* joins the server.
2.5.1. Repeat 2.1.1. & 2.1.2.
2.5.2. *Switch* selects the Black team.
2.5.3. She is assigned a starting player class.
2.5.3.1. Since the Black team already has a King, she spawns in as a Pawn.



Mockup 1: scoreboard after some players have connected

3. Players accumulate points
3.1. *Trinity* (Pawn) finds *Cypher* (Pawn) and frags him by shooting him with her weapon.
3.1.1. *Trinity* is awarded one point for fragging a player of equal class.



Mockup 2: Trinity (Pawn) fragging Cypher (Pawn)

3.2. Scenario 3.1. is repeated another two times.
3.2.1. *Trinity* now has three points banked.
3.3. *Morpheus* (King) frags *Trinity* (Pawn) by punching her with his fists.
3.3.1. *Morpheus* is not awarded any points for fragging a player of lesser class.

4. Players select player classes
4.1. *Trinity* selects a player class to spawn in as.
4.1.1. *Trinity* has three points banked, so she has enough to choose Knight (3 points), Bishop (3 points), or Pawn (free); a menu pops up upon her death and she selects Knight.

4.1.2. *Trinity* respawns as a Knight, and three points are removed from her banked points.

4.2. *Trinity* (Knight) frags *Cypher* (Pawn).

4.2.1. *Trinity* is not awarded any points for fragging a player of lesser class.

4.3. *Cypher* respawns.

4.3.1. *Cypher* has zero points, so he can only afford to be a Pawn (free); the class selection menu is not displayed upon his death.

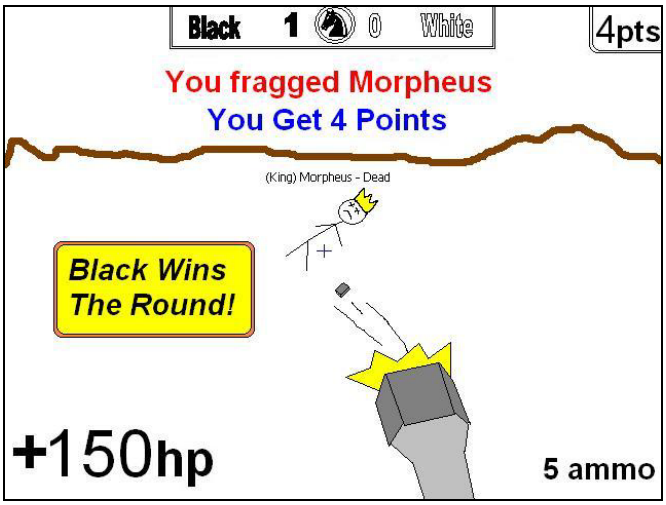
4.3.2. *Cypher* respawns as a Pawn.

5. A team frags the other team's King

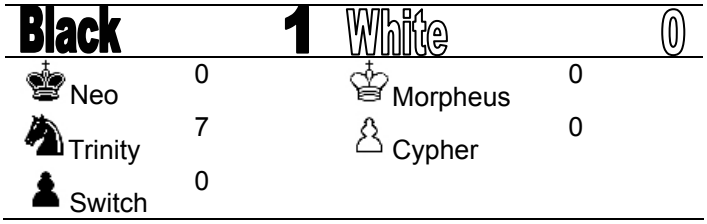
5.1. *Trinity* (Black Knight) frags *Morpheus* (White King).

5.1.1. According to **formula 1**, *Trinity* is awarded four points $\left(ceiling\left(\frac{10}{3}\right)\right)$.

5.2. Since the White King was eliminated, Black's team-score is incremented; the team scores are now Black 1, White 0; this is the end of the first round.



Mockup 3: Trinity (Knight) fragging Morpheus (King) -Black wins the round



Mockup 4: scoreboard after Black wins the first round

6. A new round begins

6.1. New Kings are assigned for both teams; this is based on the highest scoring (accumulated points) player.

6.1.1. On the Black team, *Trinity* has a score of seven and *Switch* has a score of zero, so *Trinity* respawns as the Black King for the new round and her score is reset to zero.

6.1.2. On the White team, *Cypher* is the only non-King player, he respawns as the White King for the new round and his score is reset to zero.

6.2. Every other player respawn as Pawns.

7. The match ends

7.1. During a match, scenarios 2-6 may reoccur indefinitely.

7.2. A match ends after a certain amount of time, or after a certain number of rounds have been played, or after a team reaches a certain score (these variables are determined by the server administrator).
7.2.1. The team with the highest score at the end of the match is declared the winner.
7.2.2. A new map is loaded, a new match starts, team scores and individual points are reset.

5. Objectives

Since this will be a modification of the UT2003 engine, Checkmate will be available for free download to any user with the UT2003 game. This game modification will add more value to the purchased product, by providing additional gameplay. We feel that this helps the gaming community grow and will give us valuable experience in game design; something we are all interested in.

Key criteria for success:

- Fun/unique gameplay
  Unique elements
  - team-based class-oriented gameplay
  - although player classes have been seen before, it is new to the UT2003 platform
  - taking roles of characters based on chess pieces, never seen before
  - no one has ever extended an action gaming system based on Chess
  - we chose Chess because it incorporates strategies, and is easily identifiable
  - unique weapons and skills for each class
  - unique and balanced scoring system to advance up the class hierarchy (based on value of Chess pieces)

  "Fun factor" will be evaluated by the customer
- High level of usability (easy to use)
  - Various usability tests will be performed (ie: Heuristic, Videotape, etc)

6. Architecture

Our system will be a modification of an existing game engine. This modification uses the UT2003 framework as a foundation. UT2003 uses a client-server architecture. For a given session, there can be one server hosting 0 to N clients. A client can run on the same machine as the server, but for optimal performance it is recommended to run the server on a dedicated machine.

Every machine whether a server or client, interacts with the master server, which is hosted by Epic. If there is an internet connection present when a server initializes, it will connect to the master server to advertise its presence. This allows clients to download a list of available game servers from the master server. When a client asks to connect to a game server, the master server authenticates the client to ensure they have a valid license before the connection is allowed. Interactions with the master server are not possible without an internet connection, so server advertising and client authentication do not occur on a LAN session with no internet connection.

There are various ways to administer the server. The direct method is simply by using the console that initiated the server. For cases where the console is not directly available (i.e. a dedicated server running on a remote machine), any client may login as an administrator; a client that is also running the server is automatically authenticated for administration. The server can also be configured to run an HTTP server so that it can be administered remotely through a web client. *Figure-2* illustrates how these components interact with each other.

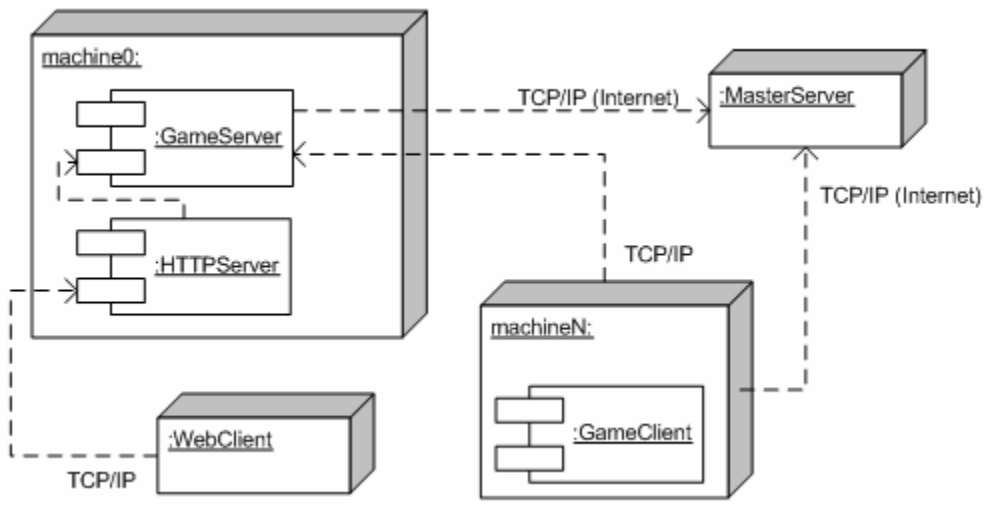


Figure 2: Deployment of a UT2003 game

What we will be modifying

We will be modifying the GameServer, and GameClient components and will rely on the master server and web clients as 3rd party components. Although the content of the HTTPServer can be modified, we currently plan to leave it unchanged.

Although the client and server run on two separate processes, they are actually programmed in tandem. The server maintains the only authoritative instance of the game, while every client maintains their own interpretation of the game. The server and clients use various techniques[3] to give the clients the best approximation of the current game state. So as we develop Checkmate, we have the ability to specify which methods of a class are valid for the server, which are valid for the client, or both.

Since Unreal is object-oriented, we will not be modifying any of the existing code. Instead, we will be subclassing relevant classes. The primary class that we will specialize is *UnrealGame.TeamGame*. This class defines the gameplay for a basic team deathmatch game. *Checkmate.CheckmateGame* will provide specialized definitions for Checkmate's game rules.

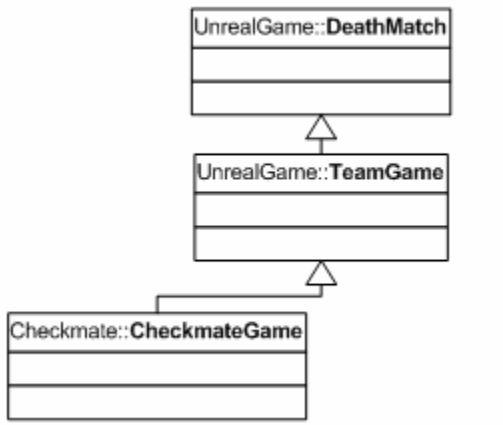


Figure 3: hierarchy of gametypes.

Unreal offers a package of classes (XInterface) that allow us to develop our own user interface. There is a large catalog of widgets available but it is also possible to define our own widgets to suit our needs.

[3] See *Appendix B* for more details on these techniques.

7. Risk Assessment

Identified risks are below:

1. Limited Support for Development Tools

Description: We will be using the tools provided by Epic Games to develop Checkmate. There is very little official documentation and support.

Action Plan: Gathering a list of resources. So far we have:

- http://udn.epicgames.com/: The Unreal Development Network.
  - This site shows the theory behind the engine.
- http://wiki.beyondunreal.com/wiki The Unreal Engine Documentation Site.
- http://www.ut2003modnetwork.com/tutorials.php
  - Some coding techniques.

2. Finding Personnel for Usability Testing

Description: We might not find enough game players for testing.

Action Plan:

- We will attempt to attain people for testing constantly throughout the development process.
- Contact online gaming website administrators for help with contacts.
- Releasing a public Beta

3. Finding a Server for Stress Testing

Description: We will need a server with high bandwidth and is capable of hosting a large number of players (32 players).

Action Plan:

- Host a Local Area Network (LAN) for testing
- Enlist the help of an online gaming website
- Release a public Beta

8. Project Plan

Roles:

Project Manager: Ashraf Ismail
Business Analyst: Bretton MacLean
Test Manager: Bretton MacLean
Build Manager: Jacques Lebrun
Architect: Jacques Lebrun
Lead Developer (game): Jacques Lebrun
Lead Developer (user interface): Ashraf Ismail
Lead Developer (test drivers): Rahwa Kelata

Outsourced:

Multimedia
Web Development

Iterations and Milestones

For this project, we will be following the deadlines given for each of the deliverables. We will be using a waterfall iterative approach, whereby we will be run through the entire iterative approach for each phase. During the first semester of the course, we will focus on completing the foundation for the game (gameplay "rules", player classes, preliminary testing, etc). For the second semester, we will implement the game itself and improve the system through play balance, player class customization, UI refinement, and stress testing.

All team members will help writing documents and presentations.

1. Project Definition:

Associated Deliverables/Milestones:

- Project Proposal 1 – January 17th, 2003
- Project Proposal 2 – January 27th, 2003
- Project Definition – January 31st, 2003
- Project Status Report – January 31st, 2003

Tasks:

Ashraf Ismail – Delegate tasks, take care of Project Plan, System description.
Jacques Lebrun – Define the architecture foundation for the system.
Bretton MacLean – Define common use cases scenarios and screen mock-ups.

2. Requirements:

Associated Deliverables/Milestones:

- Requirements Document - February 21st, 2003

Tasks:

Ashraf Ismail – Gather non-functional requirements for the document.
Jacques Lebrun – Gather functional requirements for the document.
Bretton MacLean – Get information and feedback from Adam Murray (customer).
Rahwa Keleta – Define requirements for Bots.

3. Analysis & Design:

Associated Deliverables/Milestones:

- "Hello World" version of the system – March 1st, 2003
- Analysis Report – March 7th, 2003
- Project Status Report – March 7th, 2003

Tasks:

Ashraf Ismail – Develop basics for the User Interface (in prototype form) and development programming for Player Classes.
Jacques Lebrun – Setup the development environment. This including CVS, Bugzilla problem reporting system, and automated building.
Bretton MacLean – Development programming for Player Classes.
Rahwa Keleta – Begin development of Bots.

4. Implementation:

Associated Deliverables/Milestones:

- User Interface Prototype – March 28th, 2003
- Demo of the system – April 4th, 2003
- Quality Assurance Presentation – April 4th, 2003
- Project Status Report – April 4th, 2003
- Customer Evaluation – April 4th, 2003
- Alpha version 1 – October 3rd, 2003
- Design Report – October 3rd, 2003
- Project Status Report – October 3rd, 2003

Tasks:

Ashraf Ismail – Develop User Interface prototype and then apply it to the system. Do major development for the game.
Jacques Lebrun – Coordinate development team. Do major development for the game.
Bretton MacLean – Communicate with Adam Murray (customer). Do major development for the game. Establish and maintain test cases to verify all development work.

Rahwa Keleta – Do major development for Bots.

5. Test:

Associated Deliverables/Milestones:

- Regular automated builds made available to customer – October 15th
- Completion of Bots – October 24th, 2003
- Beta version 1 – October 31st, 2003
- QA Report – October 31st, 2003
- Project Status Report – October 31st, 2003

Tasks:

Ashraf Ismail – Help with game testing. Continue with development. Coordinate defect tracking for the user interfaces.

Jacques Lebrun – Help with game testing. Continue with development. Coordinate defect tracking for the core game.

Bretton MacLean – Coordinate, setup and maintain testing. Continue with development. Receive feedback from Adam Murray.

Rahwa Keleta – Help with game testing. Setup and maintain Bots, execute predefined Bot tests. Coordinate defect tracking for the Bots.

6. Deployment:

Associated Deliverables/Milestones:

- Final Version of the system – November 21st, 2003
- Final Presentation – November 28th, 2003
- Customer Evaluation – November 28th, 2003

Tasks:

Ashraf Ismail – Make sure Final Version is completed on time.

Jacques Lebrun – Maintain list of defects and fixes.

Bretton MacLean – Communicate with Adam Murray (customer).

Rahwa Keleta – Help with final touches to the system.

Appendix A: Definitions

These are terms that will appear in this document.

Bind:	Command or set of commands that can be bound to an input unit such as a keyboard key or mouse button.
Bots:	Computer controlled players which will be used for automated testing.
FPS:	First-Person-Shooter.
Gameplay:	Interactions that happen in the game world.
Player:	User of the system's client.
Player Class:	In the context of this game, a pawn, knight, bishop, rook, queen, or king. This is not to be confused with a class in the object oriented sense.
Frag:	A term used in the context of online gaming that refers to "killing" another player. The term is less gruesome and is more appropriate since it is only a game.
Deathmatch:	A gametype where players attempt to frag each other to score.
Team Deathmatch:	A gametype where two teams attempt to frag players of the opposing team to score.

Appendix B: Overview of the Unreal Engine

The Unreal engine was originally released by Epic in 1998 under the "Unreal" title. Over the years, the engine has gone through several updates with the latest version being available through "Unreal Tournament 2003". The engine is designed to allow third party developers to create their own games either by developing a modification that plays under one of Epic's games, or by licensing the engine itself to produce a whole new game. We plan to create a modification that will be played through Unreal Tournament 2003.

UScript

Although the core engine was written in C++, Epic developed their own language called UScript for Unreal technology. Traditionally, companies have provided their modification interface in the same language as their game engine[4]. There are various reasons why Epic has gone to great efforts to develop their own programming language for their engine.

UScript is an object-oriented language that provides a high-level interface to the Unreal engine. It is a byte-code-based language similar to Java. UScript provides integrated support for various features unique to game design[5]:

- **States.** Objects in UScript have a built-in knowledge of states. This is similar to listeners in Java where a specified method is called when a given event occurs. This is a much more elegant approach than the brute-force method used in most C/C++ games. State management in such languages can become unmanageable.
- **Networking.** Unreal makes use of a powerful networking scheme that incorporates simulation, replication, and remote procedure call (RPC) techniques. These are all integrated into the UScript language.
    - Simulation allows clients to make predictions on the expected behavior of the physics of moving objects in order to conserver bandwidth. Scripts can be adjusted to optimize network usage and realism.
    - Replication allows a game server to send an approximation of the game state to the clients. Since the entire game state requires more bandwidth than is available, Unreal uses various techniques to determine the best approximation using the limited bandwidth.
    - RPCs allow server to call a function remotely on a client machine and vice versa. For example, the server may want to tell a client to play a certain sound effect; or a client might want to inform the server that (s)he wants to join the "blue" team.

UScript requires less time and code to accomplish the same thing written in C/C++.

The key drawbacks of the language are slower performance, and no low-level programming support (such as calling Windows API functions). There are some cases where low-level programming is required, in these instances UScript allows for calls to native functions similar to JNI (Java Native Interface). Where performance is critical such as rendering special effects, it is appropriate to write those associated functions in C++.

[4] i.e. Quake 1, 2, and 3 by id Software. http://www.idsoftware.com/
[5] http://unreal.epicgames.com/UnrealTechFaq.htm