# Call of the Wintermoon

Kaylen Wheeler Arvand Dorgoly

March 10, 2012

Version History

| Version Number | Edited By | Date | Comments |
|---|---|---|---|
| 0-1 | Kaylen | 21/02/2012 | Created document |
| 0-2 | Kaylen | 24/02/2012 | Started Game Overview and Game Objects and Logic sect |

## Contents

## 1 Game Overview

### 1.1 Game Summary

Play as the greatest black metal fanboy that has ever lived! Dig your way to the bottom of the cursed glacier to obtain the Blackblood Axe, the most brutal instrument of terror ever created, to begin your worldwide reign of darkness and evil. What evils await within, and will you be evil enough to overcome them?

### 1.2 Platform

The primary target platform will be the PC. Development for the PC has few barriesr compared to other platforms. Additionally, the game will be developed with relatively low technical requirements in mind, and this will open up the market to a greater number of devices. many people own PC's, and developing for this platform will make the game very widely available.

Because this game is built with the Unity engine, further platforms, such as Xbox Live Arcade or Playstation Network may be considered at a later date, as the engine allows games to be easily ported to many platforms.

# 2 Development Overview

## 2.1 Development Team

## 2.2 Development Environment

### 2.2.1 Development Hardware

### 2.2.2 Development Software

### 2.2.3 External Code

# 3 Game mechanics

## 3.1 Main Technical Requirements

## 3.2 Architecture

Since the Unity engine is being used to create and run this game, software architecture is heavily influenced by the architecture of the engine. However, there are still some important aspects of software architecture that must be decided on a per-game basis. This section will outline both Unity's software architecture and how we used that architecture to implement that of our own game.

### 3.2.1 Overview of Unity

Unity is an integrated authoring tool used for creating 3D games. By default, it provides many of the common technical elements present in most 3D games, such as high-performance graphics and physics. Graphics are implemented using either Direct3D or OpenGL, and Unity also supports Nvidia's PhysX engine. Implementation of each of these systems varies depending on the platform of deployment.

While the core of the engine has varying native implementations on different devices, scripting is implemented using Mono, the open-source implementation of the .NET Framework (originally created by Microsoft). Mono provides a software framework that allows code to run on multiple platforms. Code is compiled to the Common Language Runtime (CLR) so that code written in different high-level languages can interoperated. In Unity, three different languages are supported - UnityScript(Similar to JavaScript), Boo (Similar to Python) and C#. Because of the development team's experience, C# was chosen as the primary scripting language.

To implement scripted behaviour, the MonoBehaviour class is used. This class is part of the UnityEngine library. Multiple MonoBehavour objects can be attached to any game object, and each of them adds some behaviour to that object. The class provides several hook methods, most important of which is Update, which is called once per engine update cycle (Update cycles vary, but occur approximately 60 times per second). In addition to update functions, several event-handling functions are also provided, which can be used to detect collisions or other unexpected events that may affect the object.

### 3.2.2 Our Architecture

Using the architecture already created by Unity, our own architecture will be created to suit the purposes of our game. The primary goals of this will be to handle the enemy AI and the combat system, which are specific to this game.

The classes involved in AI control of game entities are derived from the EntityControl class, which is itself derived from MonoBehaviour. See Fig 1 for a UML diagram depicting EntityControl and other classes. EntityControl's structure is based on that of a Finite State Machine. Each instance of EntityControl contains an array of Action objects which can be indexed by any member of the EntityState enumeration. Action is a C# delegate type (similar to a function pointer) representing a void function with no parameters.

During the Start function (a function of MonoBehaviour that is run when a game object is initialized), the StateFunctions array is automatically initialized to default values. During this stage, some indices of the array are matched up with their default functions (those marked as "State Function" in the UML diagram). These functions represent states that tend to be universal across all entities. Some indices of the array correspond to more specific states, and are therefore not initialized. This is done so that exceptions will be thrown for invalid states, allowing bugs in code to be caught more effectively.

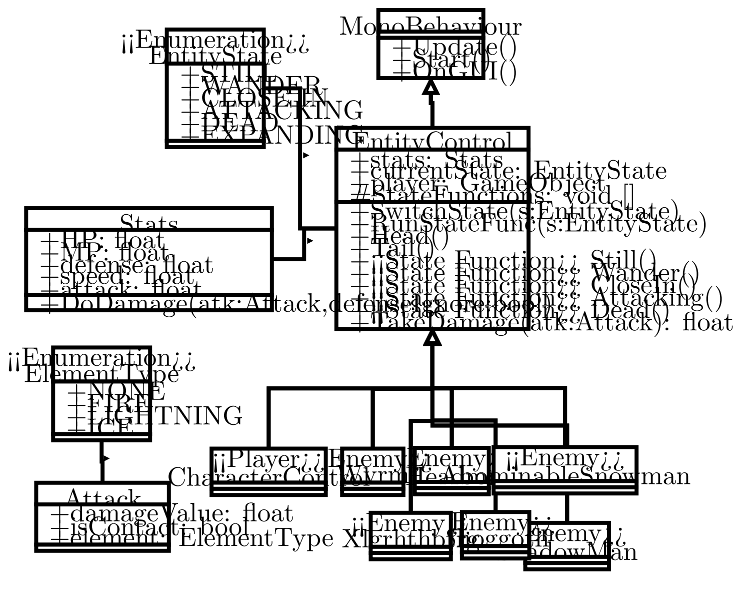

Figure 1: A UML diagram of the entity control and combat system.

The Update function in EntityControl functions primarily as a state-function dispatcher. Rather than use the Update function directly in the subclasses,

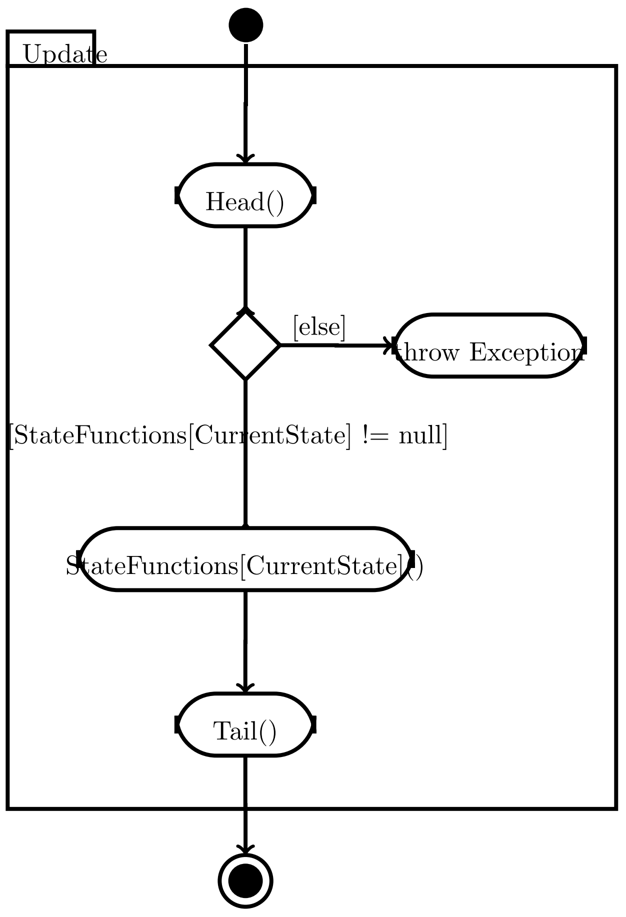
Update
Head()
[else]
Throw Exception
[StateFunctions[CurrentState] != null]
StateFunctions[CurrentState]()
Tail()

Figure 2: A UML activity diagram representing the update cycle.

hooks are provided to allow more organization. Flow of control is illustrated in Fig 2. First, the Head function is called to execute any code that may be required before the state function. Next, the current state function is fetched from the array. If there is no function for the current state, and exception is thrown and execution stops. Otherwise, the current state function is executed. Control continues to a call to the Tail method, executing any code that may be required at the end of the update cycle

The combat system has its own complexities as well. Entities are not simply destroyed when attacked, but must process the attack to compute the final amount of damage (see section 3.8.2 for more details on combat) . To store their HP, defense, and other attributes, an instance of the Stats class is used. A sequence diagram representing combat interactions is displayed in Fig. 3.

Typically, the combat system functions as follows:

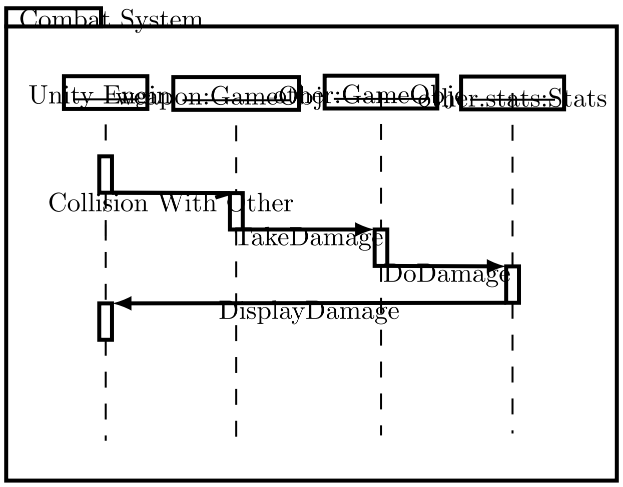

Figure 3: A sequence diagram showing combat interactions.

1. An object collides with an Entity.
2. An attack message is sent to the Entity.
3. The entity processes the attack message.
4. The attack message is then passed off to the Stats object.
5. The stats object affects the necessary changes and displays the damage value on screen.

## 3.3 Game Flow

## 3.4 Graphics

## 3.5 Audio

## 3.6 Artificial Intelligence

## 3.7 Physics

## 3.8 Game Objects and Logic

### 3.8.1 Gameplay Overview

Gameplay is restricted along the 2-dimensional XY plane, although the graphics are rendered in 3D. The basic actions can be classified as moving, attacking, and jumping. There are several extensions of these actions that will be further elaborated upon in the Abilities section.

Levels will be randomly generated, and will consist of a 2-dimensional grid of cubes, some of which may be empty. The non-empty cubes will consist of a variety of different materials (i.e. snow, ice, rock, etc.). At the beginning of the game, attacking cubes made of snow will destroy them. Throughout the game, the player will gain the ability to destroy and affect different types of terrain.

This game is a "Metroidvania" style game. This means that the levels are not arranged in a linear sequence. Rather, free exploration of the game world is encouraged. However, exploration of the game world is not completely unrestricted. Access to new areas is controlled by the acquisition of abilities that allow the player to progress.

Certain abilities are necessary to advance to new areas, and these abilities will all be required to reach the end of the game. In addition to these abilities, there will be a number of optional abilities that can be acquired through gaining experience and leveling-up.

### 3.8.2 Player and Enemy Attirbutes

Players and enemies share certain attributes that determine how they act in combat. The specific rules of combat are detailed in the Battle Mechanics section. (Player exclusive attirbutes are indicated by *.)

**HP** Determines how much damage an entity can take before it is destroyed. In the case of the player, HP is displayed in a number of discrete blocks. Each consists of 100 HP. (Similar to the energy tanks of the Metroid series.)

**MP*** Depleted when the player uses special abilities

**Attack** The amount of damage dealt by an attack.

**Defense** Resistance to attacks.

**Speed*** Affects movement and attack speed.

### 3.8.3 Combat Mechanics

When an attack connects with an entity, the attack attempts to remove HP equal to its power. Before that damage is applied, the entity's defense value is subtracted. The resulting value is clamped to a minimum of 1 HP of damage.

### 3.8.4 Abilities

(Abilities that must be obtained after beginning the game are indicated by *. Abilities necessary to reach the end of the game are indicated by **.)

**Moving** Using directional keys (A and D by default), the player can move left and right.

**Jumping** Jumping is by default accomplished with the Space key. Holding the jump key longer results in a higher jump, and tragectory can be controlled in midair.

**Wall Jumping**** A wall-jump ability is also available. The ability can be used repeatedly in order to ascend walls, but only walls of certain materials.

**Attacking and Digging** The standard attack is a simple slash with the snow shovel. This will remain the same regardless of the level of the player. Standard attacks can be aimed in any direction. This will be accomplished by moving the mouse cursor. When the player attacks, their weapon will strike in that direction.

If the attack hits a cube of terrain of any type that the player can currently destroy, that cube is destroyed.

**Special Attacks*** A number of special attacks will be available to the player. They will be able to use them if they find the necessary items. The attacks are listed below.

- **Lightning*** A bolt of lightning is fired in the direction of the cursor.
- **Fire*** The player slashes with their shovel, but the shovel is covered in flame. This results in greater damage and an increased area of effect.
- **Ice*** A temporary shield of ice is generated around the player, stopping projectiles and damaging any enemies that come into contact with it.

#### 3.8.5 Enemies

#### 3.8.6 Levels

### 3.9 Controls

### 3.10 Data Management and Flow

## 4 User Interface

### 4.1 Game Shell

### 4.2 Play Screen

## 5 Technical Risk