Software Design Document

for

HumanVsZombies, Release 1.0

Prepared by:
Kaan Akduman
Jamie Booker
Rounak Chawla
Yadira Gonzalez
Minh Pham
Sarah Yurick

EECS 393 Software Engineering
Case Western Reserve University

February 21, 2020

Table of Contents

Table of Contents **1**

Revision History **2**

Introduction **3**

1.1 Purpose 3
1.2 Scope 3
1.3 Definitions, Acronyms, and Abbreviations 3
1.4 Overview 4
1.5 References 4

Architectural Representation **4**

Architectural Goals and Constraints **5**

Use-Case Diagram **5**

4.1 User Interfaces 5
4.2 System Inputs and Outputs 7
4.3 Use-Case Realizations 8
4.3.1 Start new game 8
4.3.2 Pause game 8
4.3.3 Resume game 9
4.3.4 Reset leaderboard 9
4.3.5 Change music/sound settings 10
4.3.6 Quit game 10
4.3.7 Move player 11
4.3.8 Move object 12
4.3.9 Pick up weapon 13
4.3.10 Aim weapon 14
4.3.11 Fire weapon 14
4.3.12 Trigger new wave 15

Logical View **16**

5.1 Overview 16
5.2 Architecturally Significant Design Packages 17
5.2.1 User Interface 17
5.2.2 Main Menu 17
5.2.3 Pause Screen 17
5.2.4 Leaderboard 18

5.2.5 Settings 19
5.2.6 Map 20
5.2.7 Player 21
5.2.8 Moveable object 23
5.2.9 Weapons 23
5.2.10 Aim Arrow 25
5.2.11 Zombie 26
5.2.12 GameState 27

Risk Rank of Classes 29

Size and Performance 29

Quality 30

Inspection Report 30

Revision History

Name	**Date**	**Reason for Changes**	**Version**
Kaan, Jamie, Rounak, Yadira, Minh, Sarah	February 21, 2020	Initial draft	1.0 approved

Software Design Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of HumanVsZombies by using various architectural views to show different aspects of the video game. This document is intended to convey the significant architectural decisions that have been made for HumanVsZombies.

1.2 Scope

HumanVsZombies is intended to be a zombie shooter game that engages various levels of complexity, while still being fun and creative for the user. Some of the key features of the game are the following. The game will allow the user to interact with weapons, zombies, and the setting, which is intended to be Case Western Reserve University's campus. Additionally, the game will also feature a free for all scoring system where each wave becomes increasingly difficult (as weapons become weaker and less effective). A new wave is triggered after the player kills a certain number of zombies (the exact number is still to be determined by the game developers), and more zombies and weapons are generated at the start of a new wave. The game will end when the user runs out of health.

1.3 Definitions, Acronyms, and Abbreviations

This is a comprehensive list of all terms used in this vision document.

User/Player - A person who is able to interact with the software; they are typically playing the game.
Sprite - Graphic display of the player on the map during gameplay.
Map - The setting that the player moves around in.
Moveable object - Entity that the player can move about the map which zombies cannot move through.
Zombie - Entity that can attack the player; the player loses health when they touch a zombie.
Weapon - Entity that the player can use to attack a zombie with.
Ammo - The number of "uses" allowed for each weapon.
Wave Number - The level of the player, dependent on the number of zombies that the player has killed.
Player Score - A function of the time the player has lasted without losing all health and the number of zombies killed.
Leaderboard - Shows the highest scores obtained by players in previous games.

Player Health - Quantitative way to determine how many attacks the player can withstand.

1.4 Overview

In the following sections we outline HumanVsZombies in greater detail. We will begin by outlining the key features that will be implemented into the game. Then we will discuss the possible constraints that our game may possess as well as its quality ranges.

We will also discuss all other product requirements, platform requirements, and environment requirements in the following sections. Documentation requirements such as user tutorials and manuals will also be discussed.

Note that this document largely expands upon the requirements listed in the *Software Requirements Specification for HumanVsZombies* document [1], with some minor changes to attributes such as where certain buttons are displayed on the screen.

1.5 References

1. Akduman, et al. Software Requirements Specification. *Software Requirements Specification for HumanVsZombies*, 8-16.

2. Architectural Representation

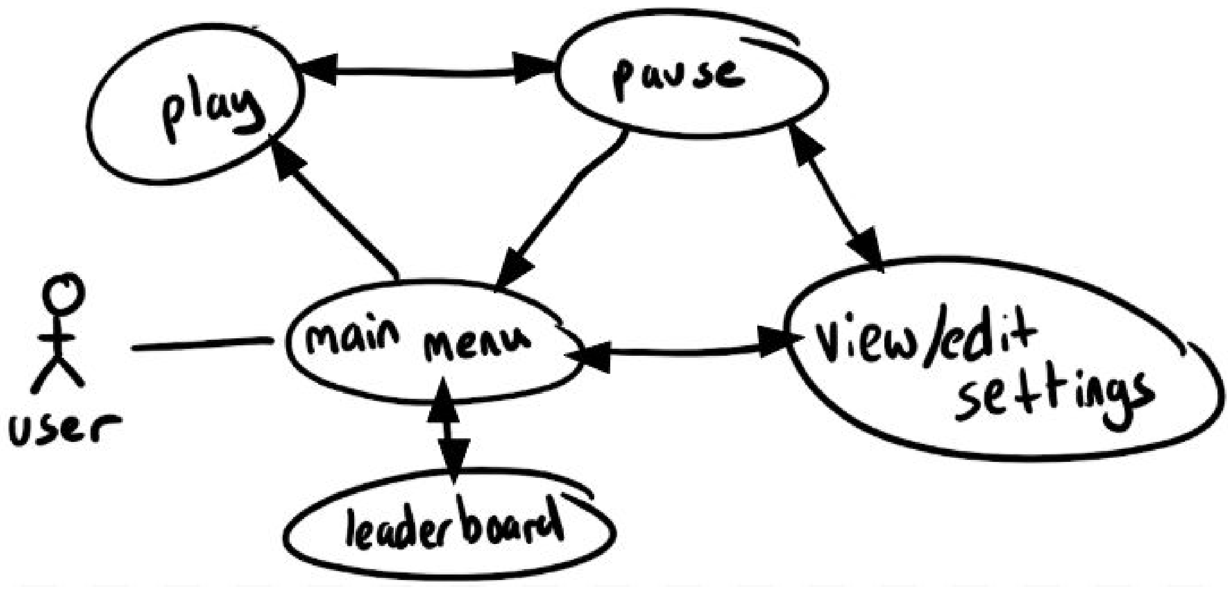

Figure 1 - Architectural Representation. Each oval is a different screen in the game. An arrow pointing from oval 1 to oval 2 indicates that screen 2 can be reached by clicking a button on screen 1.

3. Architectural Goals and Constraints

- HumanVsZombies will be able to be used on a Windows PC desktop and no mobile devices.
- HumanVsZombies will be created with an intuitive GUI.

4. Use-Case Diagram

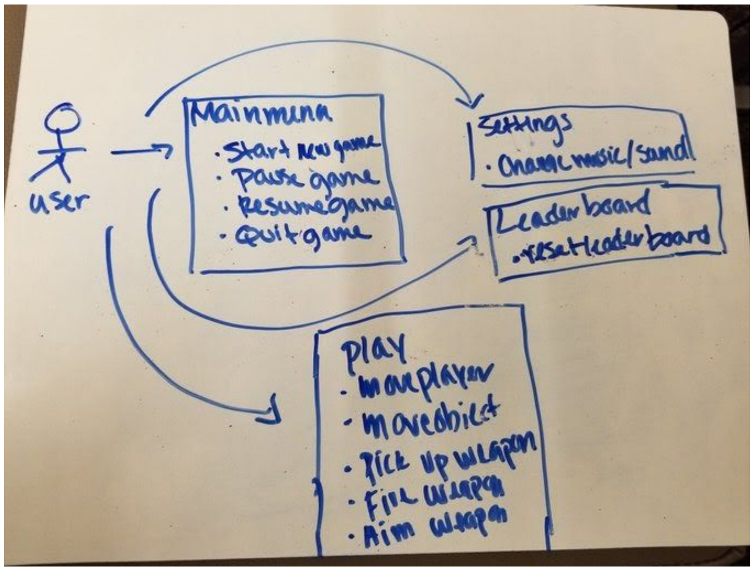

Figure 2 - Use-Case Diagram. This diagram includes the Use-Case Realizations (see 4.3) associated with each screen.

4.1 User Interfaces

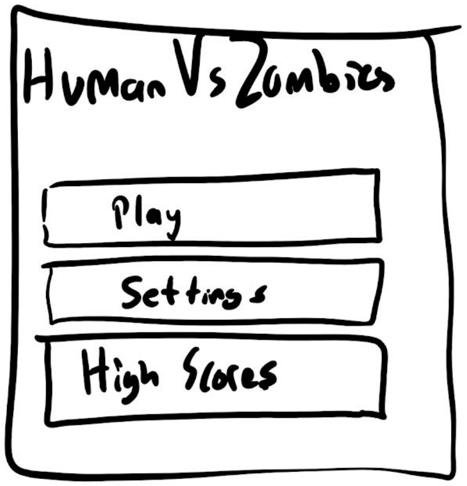

Figure 3 - Main Menu UI.

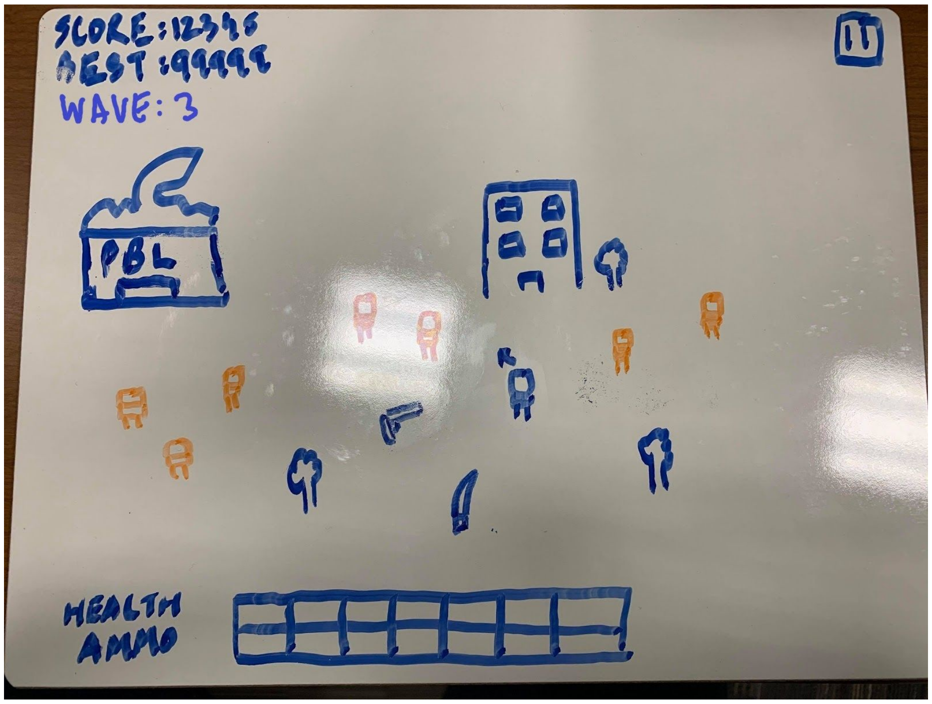

Figure 4 - Gameplay UI. Sketch of possible game map with the player (with arrow indicating the direction it is facing), zombies (in orange), scattered weapons, buildings, health and ammo bars, scores and wave number in upper left, and pause button in upper right.

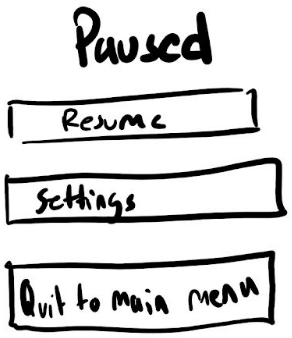

Figure 5 - Pause Screen UI.

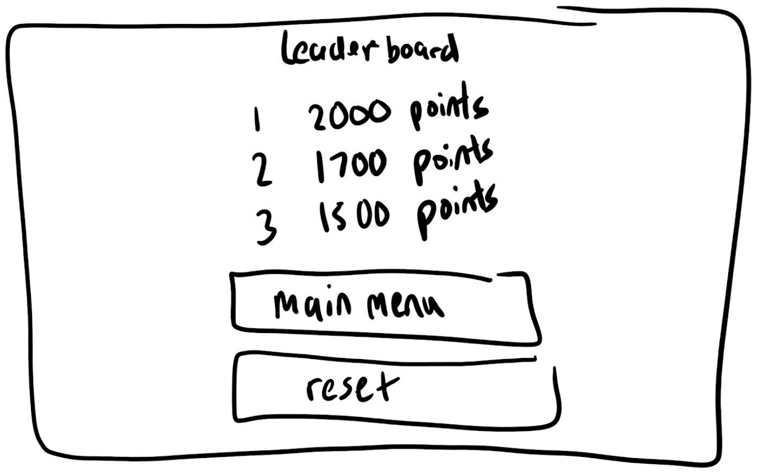

Figure 6 - Leaderboard Screen UI.

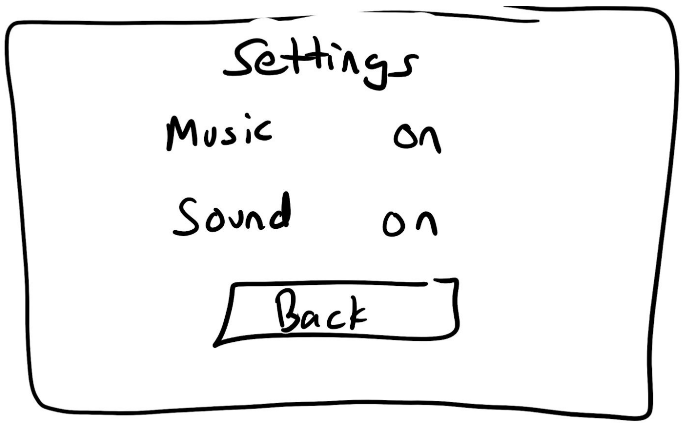

Figure 7 - Settings Screen UI.

4.2 System Inputs and Outputs

Inputs

- The system accepts input from the mouse and keyboard.

Outputs

- The system outputs player movement corresponding to the user pressing the arrow keys or the WASD keys.
- The system outputs an updated image of the player holding a weapon (indicating that the player can use the weapon) after they move to touch that weapon.
- The system outputs an arrow indicating the direction that the weapon is pointing on the screen, depending on where the mouse is located on the screen.
- The system outputs firing a weapon when the user clicks the mouse.
- The system outputs newly generated zombies and weapons at the start of a new wave.

- The system outputs the score and highscores at the end of a game on the screen.

4.3 Use-Case Realizations

4.3.1 Start new game

- **Brief Description**
  - In this use case the user starts playing a new game of HumanVsZombies.
- **Actors**
  - The user.
- **Preconditions**
  - The user is on the "Main Menu" page.
- **Postconditions**
  - The user has started a new game, meaning that the gameplay screen, which includes the map, the player which the user can control, zombies and weapons on the map, and displays of scoring and health information.
- **Normal course scenario**
  - This use case begins with the user on the main menu page. The user clicks the "Play" button.
  - After the user clicks "Play," a short (10-20 seconds) story video sequence is played, which provides a brief outline of the zombie apocalypse in the game. This step is low priority.
  - After the video sequence, the user's player is placed on the map for a new game with an initial score of zero. Zombies, weapons, and moveable objects are generated randomly on the map.
  - End of use case.

4.3.2 Pause game

- **Brief Description**
  - In this use case the user pauses the game that they are currently playing.
- **Actors**
  - The user.
- **Preconditions**
  - The user is currently playing the game.
- **Postconditions**
  - The game is paused, meaning that neither the player nor the zombies are moving and the timer has paused.
- **Normal course scenario**
  - This use case begins while the user is playing the game. In the upper right corner, they press the pause button.
  - The pause screen, which consists of the options "Resume," "Settings," and "Quit to Main Menu" is displayed.

  - The internal timer is paused.
  - The zombies are frozen in place in the gamestate.
  - If the user presses any of the arrow keys or WASD keys, or clicks the mouse, nothing will happen; the player cannot move or fire their weapon.
  - End of use case.

4.3.3 Resume game

- **Brief Description**
  - In this use case the user unpauses the game that they are currently playing.
- **Actors**
  - The user.
- **Preconditions**
  - The user has paused the game and is on the pause screen.
- **Postconditions**
  - The user has resumed playing the game that they had paused.
- **Normal course scenario**
  - This use case begins with the user on the pause screen. They click the "Resume" button.
  - The internal timer for the game is unpaused and resumes.
  - The zombies are unfrozen and can move around again.
  - The user is able to move the player around and fire weapons again.
  - End of use case.

4.3.4 Reset leaderboard

- **Brief Description**
  - HumanVsZombie keeps track of the top 3 scores from previous games played. The user can choose to reset these scores back to zero.
- **Actors**
  - The user.
- **Preconditions**
  - The user is on the leaderboard page, which can be reached by clicking the "High Scores" button on the main menu.
- **Postconditions**
  - All 3 high scores have been reset to zero.
- **Normal course scenario**
  - This use case begins with the user on the leaderboard screen. They click the "Reset" button.
  - The variables which store the top 3 high scores from previous games are reset to zero. This change is displayed on the screen.
  - End of use case.

4.3.5 Change music/sound settings

- **Brief Description**
  - HumanVsZombies includes background music and sound effects within the game. These can be turned off or on as the user desires.
- **Actors**
  - The user.
- **Preconditions**
  - The user is on the settings page, which can be reached either by clicking the "Settings" button from the main menu screen or the "Settings" button from the pause screen.
  - The system has kept track of whether to play music and/or sound effects during gameplay.
- **Postconditions**
  - The music toggle button is updated accordingly and the music either starts or stops playing.
  - The sound effects toggle button is updated accordingly, and the next time the player starts or resumes the game there either are or are not sound effects.
- **Normal course scenario**
  - This use case begins with the user on the settings screen.
  - If the user clicks the toggle button for the music, then the music switches to "off" if it was "on" before, and vice versa to "on" if it was "off" before.
    - This change is displayed on the screen as the toggle button changes.
    - The music immediately begins or stops playing as the user clicks the button.
  - If the user clicks the toggle button for the sound effects, then the sound effects switch to "off" if they were "on" before, and vice versa to "on" if they were "off" before.
    - This change is displayed on the screen as the toggle button changes.
    - When the player starts or resumes the game, the existence of sound effects corresponds to the toggle button's state.
  - End of use case.

4.3.6 Quit game

- **Brief Description**
  - In this use case the user can choose to terminate the game they are currently playing.
- **Actors**
  - The user.
- **Preconditions**
  - The user is on the pause screen.
- **Postconditions**

- The user has been returned to the main menu.

- **Normal course scenario**
  - This use case begins with the user on the pause screen. The user clicks the “Quit to Main Menu” button.
  - The system checks the user’s current score against the leaderboard scores. If the score for the game is a new high score, the leaderboard values are adjusted accordingly.
  - The gameplay terminates; the score and internal timer are reset to zero.
  - The main menu screen is displayed to the user.
  - End of use case.

4.3.7 Move player

- **Brief Description**
  - In this use case the user moves the player to any location on the map by pressing either the arrow keys or WASD keys on the keyboard.
- **Actors**
  - User pressing their keyboard, Player, Map
- **Preconditions**
  - The system has kept track of where the player is currently located on the map.
- **Postconditions**
  - The player has moved to a different location on the map, which is updated in the system.
- **Normal course scenario**
  - This use case begins when the user presses and holds one of the arrow keys or WASD keys.
  - The user sees the player move across the screen in the direction corresponding to the key pressed.
    - Upper arrow/W key allows the player to move upward.
    - Left arrow/A key allows the player to move left.
    - Down arrow/S key allows the player to move downward.
    - Right arrow/D key allows the player to move right.
  - While the user is pressing the key, the system is constantly updating and keeping track of where the player is located on the map.
  - When the user releases the key, the player stops moving on the map.
  - End of use case.
- **Alternative course 1: The user tries to move the player outside of the map.**
  - This use case begins when the user presses and holds one of the arrow keys or WASD keys.

- However, the player is located on the edge of the screen, meaning that the map terminates in the direction in which the user is attempting to move the player.
- The system recognizes that the user is on a boundary of the map and does not allow the player to move in that direction.
- The system does not update the player's location on the map since the player did not move.
- End of use case.

4.3.8 Move object

- **Brief Description**
    - There are objects on the map which the user can move by coming into contact with them.
- **Actors**
    - User pressing their keyboard, Player, Map, Moveable object
- **Preconditions**
    - The system has kept track of where the moveable object is currently located on the map.
    - The player is next to the moveable object.
- **Postconditions**
    - The moveable object has moved to a different location on the map, which is updated in the system.
- **Normal course scenario**
    - This use case begins when the user presses and holds one of the arrow keys or WASD keys while they are facing a moveable object.
    - The user sees the player and object move across the screen in the direction corresponding to the key pressed.
        - Upper arrow/W key allows the player to move the object upward.
        - Left arrow/A key allows the player to move the object to the left.
        - Down arrow/S key allows the player to move the object downward.
        - Right arrow/D key allows the player to move the object to the right.
    - While the user is pressing the key, the system is constantly updating and keeping track of where the object is located on the map.
    - When the user releases the key, the object stops moving on the map.
    - End of use case.
- **Alternative course 1: The user tries to move the object outside of the map.**

- This use case begins when the user presses and holds one of the arrow keys or WASD keys while they are facing a moveable object.
- However, the moveable object is located on the edge of the screen, meaning that the map terminates in the direction in which the user is attempting to move the object.
- The system recognizes that the object is on a boundary of the map and does not allow the player to move the object in that direction.
- The system does not update the moveable object's location on the map since the object did not move.
- End of use case.

4.3.9 Pick up weapon

- **Brief Description**
  - The player is able to pick up weapons, which are scattered on the map, to use to kill zombies.
- **Actors**
  - Player, Map, Weapon
- **Preconditions**
  - There is a weapon on the map.
  - The player is not currently holding a weapon.
- **Postconditions**
  - The player's sprite has been updated to display the player holding the weapon.
  - The weapon can be aimed and fired by the user.
  - A static ammo bar for the weapon is displayed at the bottom of the map
- **Normal course scenario**
  - This use case begins when the user moves the player to come in contact with a weapon on the map.
  - When the player touches the weapon, their sprite is updated to display the player holding that weapon.
  - A bar displaying the weapon's remaining ammo is displayed on the bottom of the screen. The weapon starts out with full ammo.
  - The weapon is taken off of the map.
  - The system records that the player is now holding a weapon, which the player can aim and fire.
  - End of use case.
- **Alternative course 1: The player is already holding a weapon.**
  - This use case begins when the user moves the player to come in contact with a weapon on the map.
  - When the player touches the weapon, the system recognizes that the player is already holding a weapon.
  - The player retains the weapon they were holding before.
  - The weapon the player attempted to pick up remains on the map.

- ○ End of use case.

4.3.10 Aim weapon

- **Brief Description**
  - The player is able to aim their weapon corresponding to where the mouse is located on the screen. The direction the player is aiming the weapon is indicated by an arrow which circles the player.
- **Actors**
  - Player, Weapon
- **Preconditions**
  - The player is holding a weapon.
- **Postconditions**
  - The player’s arrow is pointing in the direction of the mouse’s location on the screen.
- **Normal course scenario**
  - This use case begins with the player holding a weapon. The player moves the mouse to somewhere on the gameplay screen.
  - The system keeps track of the mouse’s location and the arrow which circles the player rotates to point in the same direction as the mouse.
  - If the user clicks the mouse, the weapon will fire in the direction that the arrow is pointing.
  - End of use case.
- **Alternative course 1: The player is not holding a weapon.**
  - This use case begins with the player on the map during gameplay. The player is not holding a weapon.
  - The system does not keep track of the mouse when the player is not holding a weapon.
  - There is no arrow circling the player when the player is not holding a weapon.
  - End of use case.
- **Alternative course 2: The user hovers the mouse over the player.**
  - This use case begins with the player holding a weapon. The player moves the mouse to hover over the player.
  - Since the player cannot aim at themselves, the system simply maintains the arrow pointing in its most recent valid direction.
  - If the user clicks the mouse, the weapon will fire in the direction that the arrow is pointing.
  - If the user moves the arrow off of the player, then the direction of the arrow is updated as specified in the normal case scenario.
  - End of use case.

4.3.11 Fire weapon

- **Brief Description**

- The player is able to fire their weapon in the direction that the arrow which circles the player is pointing. If this direction is within range of a zombie, then firing it at the zombie will damage or kill the zombie.

- **Actors**
  - Player, Weapon, Zombie
- **Preconditions**
  - The player is holding a weapon and the arrow which encircles the player is pointing in some direction.
- **Postconditions**
  - The player has fired the weapon, decrementing the ammo and possibly damaging or killing a zombie.
- **Normal course scenario**
  - This use case begins with the player holding a weapon and the mouse located in the direction in which the player intends to fire. The user clicks the mouse.
  - An animation of the weapon firing or being used is displayed.
  - The ammo count, which is displayed at the bottom of the screen, decrements.
  - If a zombie is within range and the direction that the player is pointing the mouse, the zombie endures damage or is killed.
    - The distance which is considered to be “within range” is left to the choice of the game developers and may also depend on which weapon the player is holding.
    - The capacity to which each weapon damages a zombie is left to the choice of the game developers.
    - If a zombie is killed, then the system’s counter for the number of zombies killed increments by 1.
  - End of use case.
- **Alternative course 1: The player is not holding a weapon.**
  - This use case begins with the player on the map during gameplay. The player is not holding a weapon.
  - The player clicks the mouse.
  - Since the player is not holding a weapon, nothing happens on the screen and the game resumes normally.
  - End of use case.

4.3.12 Trigger new wave

- **Brief Description**
  - After the player has killed a certain number of zombies, a new wave begins, spawning more zombies and weapons on the map.
- **Actors**
  - GameState
- **Preconditions**

- The system has kept track of the total number of zombies killed in the current game.
- The user has killed some specified number of zombies (the exact number is left to the decision of the game developers).

- **Postconditions**
  - A new wave has begun, meaning that more zombies and weapons were generated randomly on the map.
- **Normal course scenario**
  - This use case begins when the player kills the specified number of zombies that they have to kill to trigger a new wave.
  - The wave number is updated on the upper left of the gameplay screen.
  - New zombies are generated at random locations on the map. They are not generated at the same location as the player's current location.
  - New weapons are generated at random locations on the map. The weapons are worse (inflict less damage) than the weapons generated in the previous wave. They are not generated at the same location as the player's current location.

5. Logical View

5.1 Overview

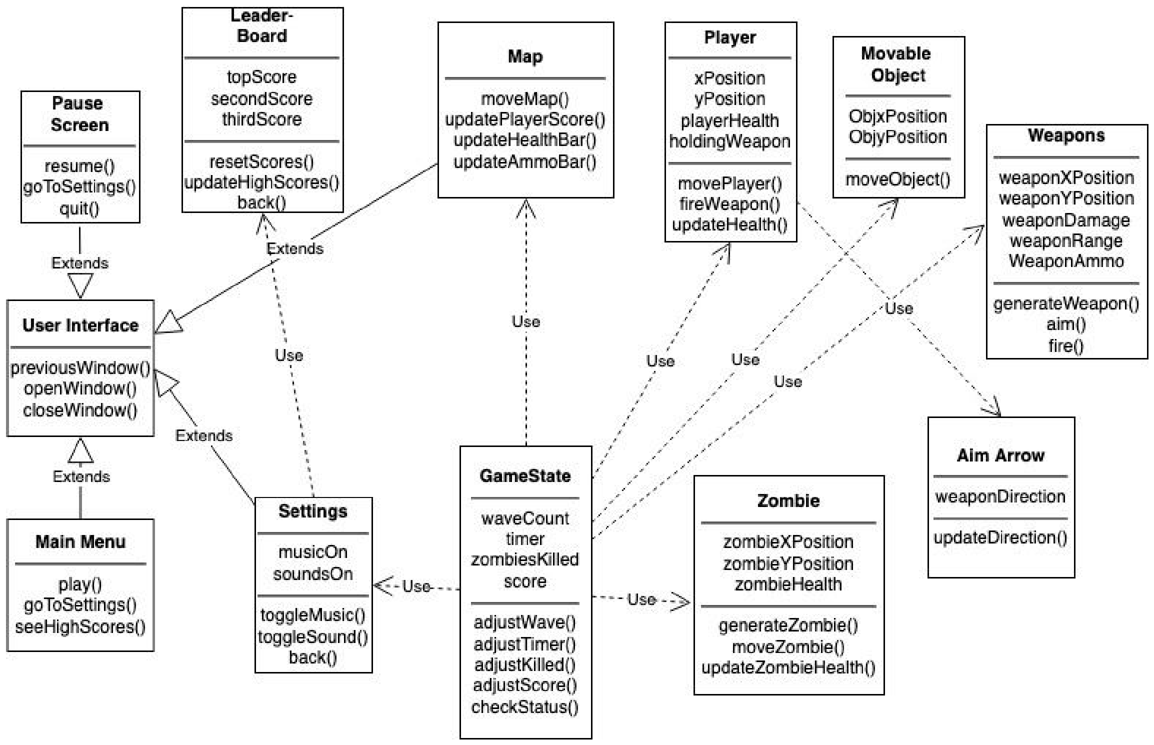

Figure 8 - System Class Design. UML diagram.

5.2 Architecturally Significant Design Packages

5.2.1 User Interface

a. **Brief description:** Provides basic functionality for users, such as open, close, and go back.

b. **Methods:**

Access	**Return**	**Name**	**Description**
Public	Void	previousWindow()	Holds the previous window opened before the current one.
Public	Void	openWindow()	Opens a new user interface window, allowing users to make changes.
Public	Void	closeWindow()	Closes a currently opened window.

5.2.2 Main Menu

a. **Brief description:** The user sees the main menu when first opening the game application and when a game ends.

b. **Methods:**

Access	**Return**	**Name**	**Description**
Public	Void	play()	Prompts the user to play the game in the main interface.
Public	Void	goToSettings()	Allows the user to go to the settings screen.
Public	Void	seeHighScores()	Allows the user to go to the leaderboard screen.

5.2.3 Pause Screen

a. **Brief description:** The user sees the pause screen when they click the pause button at the top right of the screen during gameplay.

b. **Methods:**

Access	**Return**	**Name**	**Description**
Public	Void	resume()	Allows the user to continue the game after being paused.
Public	Void	goToSettings()	Allows the user to go to the settings screen.
Public	Void	quit()	Brings the user back to the opening menu, erasing all progress saved after checking if the user achieved a new high score.

5.2.4 Leaderboard

a. **Brief description:** This screen displays the top 3 scores achieved by the user.
b. **Attributes:**

Type	**Access**	**Name**	**Description**
Integer	Public	topScore	This attribute stores the highest score achieved across all gameplays.
Integer	Public	secondScore	This attribute stores the second highest score achieved across all gameplays.
Integer	Public	thirdScore	This attribute stores the third highest score achieved across all gameplays.

c. **Methods:**

Access	**Return**	**Name**	**Description**
Public	Void	resetScores()	Resets topScore, secondScore, and

			thirdScore to zero.
Public	Void	updateHighScores()	If the user achieves a new top 3 score, then the attributes topScore, secondScore, and thirdScore are updated accordingly. For example, if the user achieves a new topScore, thirdScore is updated to secondScore's value, secondScore is updated to topScore's value, and topScore is updated to the new high score.
Public	Void	back()	Goes back to the previous screen that the user was on, which was the main menu.

5.2.5 Settings

a. **Brief description:** The user can go to the settings screen either from the main menu or the pause screen and change music and sound effects settings.

b. **Attributes:**

Type	**Access**	**Name**	**Description**
Boolean	Public	musicOn	This attribute keeps track of whether or not the music is playing.
Boolean	Public	soundsOn	This attribute keeps track of whether or not there is sound effects during gameplay.

c. Methods:

Access	**Return**	**Name**	**Description**
Public	Void	toggleMusic()	Allows the user to change the background music of the game to be on or off.
Public	Void	toggleSound()	Allows the user to change the in-game sound effects to be on or off.
Public	Void	back()	Goes back to the previous screen that the user was on, either the main menu or the pause screen.

5.2.6 Map

a. Brief description: Provides the main interface of the game, where the progress happens. The user will be presented with a bird's eye, two-dimensional top view of the game map.

b. Attributes:

Type	**Access**	**Name**	**Description**
Integer	Public	xRange	This attribute stores how large the map is in the x-direction.
Integer	Public	yRange	This attribute stores how large the map is in the y-direction.

c. Methods:

Access	**Return**	**Name**	**Description**
Private	Void	moveMap()	Displays more areas of the map in the moving direction,

			while no longer displaying passed parts.
Private	Void	updateScoreDisplay()	Displays the player's score in the upper left corner according to the score attribute in GameState.
Private	Void	updateHealthBar()	Updates the player's health status bar on the bottom of the screen, based on the playerHealth attribute in Player class.
Private	Void	updateAmmoBar()	Updates the ammunition bar on the bottom of the screen, based on the weaponAmmo attribute in Weapon class.

5.2.7 Player

a. **Brief description:** Stores information about the player during gameplay.
b. **Attributes:**

Type	**Access**	**Name**	**Description**
Integer	Public	xPosition	This attribute stores the current x-coordinate position of the player on the map.
Integer	Public	yPosition	This attribute stores the current y-coordinate position of the player on the

			map.
Integer	Public	playerHealth	This attribute stores the player's current health.
String	Public	holdingWeapon	This attribute stores which weapon the player is holding. The attribute is NULL when the player is not holding a weapon.

c. Methods:

Access	**Return**	**Name**	**Description**
Public	Void	movePlayer()	This method keeps updating the player's xPosition and yPosition as the user presses the arrow keys or WASD keys. The user can also see the player move across the map as they are pressing the arrow keys or WASD keys.
Public	Void	fireWeapon()	After checking that the player is holding a weapon with sufficient ammo, the player clicks the left mouse and sees an animation of the player using the weapon.
Public	Void	updateHealth()	Updates the attribute playerHealth, decrementing it when the player comes in

			contact with a zombie.

5.2.8 Moveable object

a. **Brief description:** There are objects on the map which the player can move by coming into contact with them. These objects can be used to block zombies.

b. **Attributes:**

Type	**Access**	**Name**	**Description**
Integer	Public	ObjxPosition	This attribute stores the current x-coordinate position of the object on the map.
Integer	Public	ObjyPosition	This attribute stores the current y-coordinate position of the object on the map.

c. **Methods:**

Access	**Return**	**Name**	**Description**
Public	Void	moveObject()	This method keeps updating the object's ObjxPosition and ObjyPosition as the user moves the object. The user can also see the object move across the map with the player as they are pressing the arrow keys or WASD keys.

5.2.9 Weapons

a. **Brief description:** The player can pick up weapons which can be used to harm or kill zombies.

b. Attributes:

Type	**Access**	**Name**	**Description**
Integer	Public	weaponXPosition	This attribute stores the current x-coordinate position of the weapon when it is on the map. If the player is holding the weapon, this attribute is set to NULL.
Integer	Public	weaponYPosition	This attribute stores the current y-coordinate position of the weapon when it is on the map. If the player is holding the weapon, this attribute is set to NULL.
Integer	Public	weaponDamage	This attribute stores how much damage is done to a zombie's health when the weapon is used on the zombie 1 time.
Integer	Public	weaponRange	This attribute stores how far away a zombie can be from the player for the weapon to be effective.
Integer	Public	weaponAmmo	This attribute stores how many more times the weapon can be used.

c. Methods:

Access	Return	Name	Description
Private	Void	generateWeapon( )	Randomly generates the weapons' locations on the map at the beginning of a wave.
Public	Void	aim()	Aims the firing direction according to where the mouse is pointing. An arrow indicating the direction will be displayed.
Public	Void	fire()	Weapon fires when the mouse is clicked in the direction that the aim arrow is pointing. When the player fires the weapon, weaponAmmo is decremented by 1.

5.2.10 Aim Arrow

a. **Brief description:** When the player is holding a weapon, there is an arrow which circles the player which points in the direction of wherever the mouse is located on the screen.

b. **Attributes:**

Type	Access	Name	Description
Double	Public	weaponDirection	This attribute stores the current direction that the user is pointing the weapon. A real value from 0 to 359.999.

c. **Methods:**

Access	Return	Name	Description
Public	Void	updateDirection()	When the player is holding a weapon, an aim arrow which circles

			the player rotates according to where the mouse is located on the screen. The weaponDirection attribute is updated accordingly.

5.2.11 Zombie

a. **Brief description:** Zombies are the antagonists of the game. A player loses health when they come in contact with a zombie.

b. **Attributes:**

Type	**Access**	**Name**	**Description**
Integer	Public	zombieXPosition	This attribute stores the current x-coordinate position of the zombie when it is on the map. If the zombie is killed, this attribute is set to NULL.
Integer	Public	zombieYPosition	This attribute stores the current y-position of the zombie when it is on the map. If the zombie is killed, this attribute is set to NULL.
Integer	Public	zombieHealth	This attribute stores the health of the zombie.

c. **Methods:**

Access	**Return**	**Name**	**Description**
Private	Void	generateZombie()	Randomly generates the zombies'

			locations. The higher the wave number, the more zombies that are generated.
Private	Void	moveZombie()	The zombies move around the map; this method updates the zombieXPosition and zombieYPosition attributes.
Public	Void	updateZombieHealth()	When a weapon is fired within range and in the direction of a zombie, the zombie's health decrements by the weapon's weaponDamage attribute.

5.2.12 GameState

a. **Brief description:** This class stores information about the state of the game.
b. **Attributes:**

Type	**Access**	**Name**	**Description**
Integer	Public	waveCount	This attribute stores the current wave that the player has achieved in the game.
Integer	Private	timer	This attributes stores, in seconds, how long the user has been playing the game.

Integer	Private	zombiesKilled	This attribute stores how many total zombies have been killed in the game.
Integer	Public	score	This attribute stores the user's score as a function of gameplay time and zombiesKilled.

c. Methods:

Private	Void	adjustWave()	After a sufficient number of zombies have been killed, waveCount is incremented by 1 and a new wave begins. This triggers the generateWeapon() and generateZombie() methods.
Private	Void	adjustTimer()	Increments the timer by 1 every second that passes.
Private	Void	adjustKilled()	Checks for when a zombie runs out of health and increments zombiesKilled accordingly.
Private	Void	adjustScore()	Adjusts the score attribute as a function of gameplay time and zombiesKilled. The exact function used is left to the decision of the game developers.
Private	Boolean	checkStatus()	Based on updateHealth, if the return value is 0,

			displays that the user has lost and quits. Otherwise, continue.

6. Risk Rank of Classes

User Interface	High
GameState	
Map	
Player	
Aim Arrow	
Weapons	
Zombie	
Main Menu	
Pause Screen	
Moveable object	
Leaderboard	
Settings	Low

7. Size and Performance

- Bullets or some other form of ammunition should come out of ranged weapons within 0.02 seconds after clicking the attack key.
- Melee weapons should begin swinging within 0.02 seconds after clicking the attack key.
- Zombies should lose health 0.02 seconds after the melee weapon or ammunition comes in contact with it.
- If a zombie's health reaches 0, it should die within 0.02 seconds.
- The game will accurately save all high scores.

8. Quality

In order to maintain an accurate depiction of gameplay status, the system will ensure that all attributes related to calculating the score are properly recorded and updated as necessary.

9. Inspection Report

Team Member	**Findings**	**Resolution**
Kaan Akduman	Buttons described in Use-Case Realizations and Logical View sections were not in the User Interfaces section.	Updated the User Interfaces to match the other sections.
Jamie Booker	Settings was missing a "Back" button. Table of Contents out of order.	Added a back button to settings. Rearranged table.
Rounak Chawla	Missing "Overview" section in Logical View.	Added UML diagram.
Yadira Gonzalez	Architectural Representation and Use-Case Diagrams were vague and unclear.	Remade diagrams.
Minh Pham	Found vague class descriptions.	Added more detail.
Sarah Yurick	The wave number was missing from the Gameplay user interface in section 4.1 Found inconsistencies between diagrams and written descriptions.	Edited the photo to include wave number. Reported findings to team members to adjust accordingly.