3ly-Zium
for UTM CSCI 352

Colin Weatherly, Blade Johnson

Abstract

The project is a video game in the 2D platformer genre. The player is made to run through a level, and upon each successful completion, the level is changed to increase the difficulty. The intended target is people that play video games, primarily those that stream games as live reactions mesh well with high difficulties.

1. Introduction

This project is a video game in the genre of 2D platformer. The goal of the game is to reach the exit of the level that the player character is placed in. The level's layout will progressively get more complex with each clear, along with adding new hazards that increase the difficulty further. The penalty for failing a level is to be sent back to the beginning of said level, allowing the player to learn the level with every attempt.

The targeted audience for this project is people in the 18-34 age group who play video games on a regular basis. Specifically, those in the age group that livestream games. Due to the increasingly difficult challenge that the game provides, having live reactions of attempts to beat the game will provide content for the targeted audience.

1.1. Background

A 2D platformer is a game genre set on a 2D plane where the player controlled character must move and jump to avoid obstacles and reach the end.

We decided to make a platforming game because of our love for games in general and our shared interest in platformers. Because of this, we feel we can properly create one.

1.2. Impacts

This project is not meant to impact the world in some way. The point of a game is to provide entertainment, which will only impact a portion of the population if it were to gain traction within the community. At the current time, there is no plan for a meaningful story in the game, so the impact will be purely driven by the quality of the gameplay.

1.3. Challenges

The main challenges will most likely be coding the physics of the player character; developing new mechanics instead of being purely moving and jumping; keeping the game fun despite the difficulty.

2. Scope

This platforming game is considered done once three levels of the game are playable from start to finish with minimal bugs or exploits. The player should be able to get through these stages without too much unnecessary frustration with either the difficulty of these stages or the control of the player character. The player character should be fully implemented with all moves intact (moving, jumping) and all hazards(such as spikes) should fail the level upon contact with the player character. As a stretch goal, we want to include more levels with new hazards. An additional stretch goal is the addition of extra moves(i.e walljumping, sliding) the player character requires to progress in more difficult levels. An additional stretch goal is scaling the window and the game assets to fit more than one resolution.

2.1. Requirements

2.1.1. Functional.

- User inputs via designated keys are read – the application should register the inputs and move the player character accordingly
- Selecting Start places the user in the first level – Unlike the level select which allows the user to choose a level, pressing the start button will immediately bring them to the first level
- Level fails upon contact with a hazard – should the player character run into a hazard(i.e spike), level is failed and character is returned to the starting point of the same level

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Start New Game	User	Low	1
2	Level Select	User	Low	1
3	Character Jump	User	Medium	1
4	Character Dash	User	Medium	1

TABLE 1. USE CASE TABLE

2.1.2. Non-Functional.

- The game correctly displays in a 1280x720 resolution window
- The game runs at a smooth framerate, preferably 30 frames per second
- User's inputs have little to no delay, preferably under half a second

2.2. Use Cases

These are cases in which the user will interact with the program's user interface. The use cases can be seen in Table 1.

Use Case Number:	1
Use Case Name:	Start New Game
Description:	The player decides to begin the game from the beginning (Level 1). They will click on the "Start" button (see Figure 2). This will load the first level of the game (see Figure 1).

Process flow:

1) Player starts the program, which begins with the main menu loaded.
2) The player left-clicks the "Start" option on the menu.
3) The game state is updated to gameplay, and the first level is loaded.

Termination Outcome: Gameplay has begun in level 1.

Use Case Number:	2
Use Case Name:	Level Select
Description:	The player is on the main menu. The player wishes to choose a specific level in the game to play. They will left-click the "Level Select" button on the main menu, and the screen will update to show a list of levels in vertical descending order. The player then left-click a button corresponding to a specific level, and the game will load the corresponding level (see Figure 3).

Process flow:

1) Player starts the program, which begins with the main menu loaded.
2) The player left-clicks the "Level Select" option on the main menu.
3) The level select screen is loaded, showing each level vertically in numerically descending order.
4) The player left-clicks the option corresponding to their desired level.
5) The game state is updated to gameplay, and the corresponding level loads.

Termination Outcome: Gameplay has begun in the player's selected level.

Use Case Number:	3
Use Case Name:	Character Jump
Description:	The player is in gameplay. They wish to make the player character jump (in order to avoid hazards such as spikes). The player then hits the 'Z' key. The player character will then leap upward. The character will gradually rise up off the ground for a few seconds, then gradually fall back to the ground.

Process flow:

1) Player is in a stage of gameplay.
2) The player presses the 'Z' key.
3) The player character then leaps off the ground into the air.
4) After a few seconds of gradually rising upward, the player character then gradually falls until hitting the ground.

Termination Outcome: The player character has completed a full jump.

Use Case Number:	4
Use Case Name:	Character Dash
Description:	The player is in gameplay. They wish to make a character quickly travel horizontally. The player then hits the 'X' key. The player will then gain a burst of speed in the direction they are moving. This can also be used during a jump.

Process flow:

1) Player is in a stage of gameplay.
2) The player presses the 'X' key.
3) The player's movement speed is doubled in the direction of movement(left, right) for a second.

Termination Outcome: The player has completed a full dash.

2.3. Interface Mockups

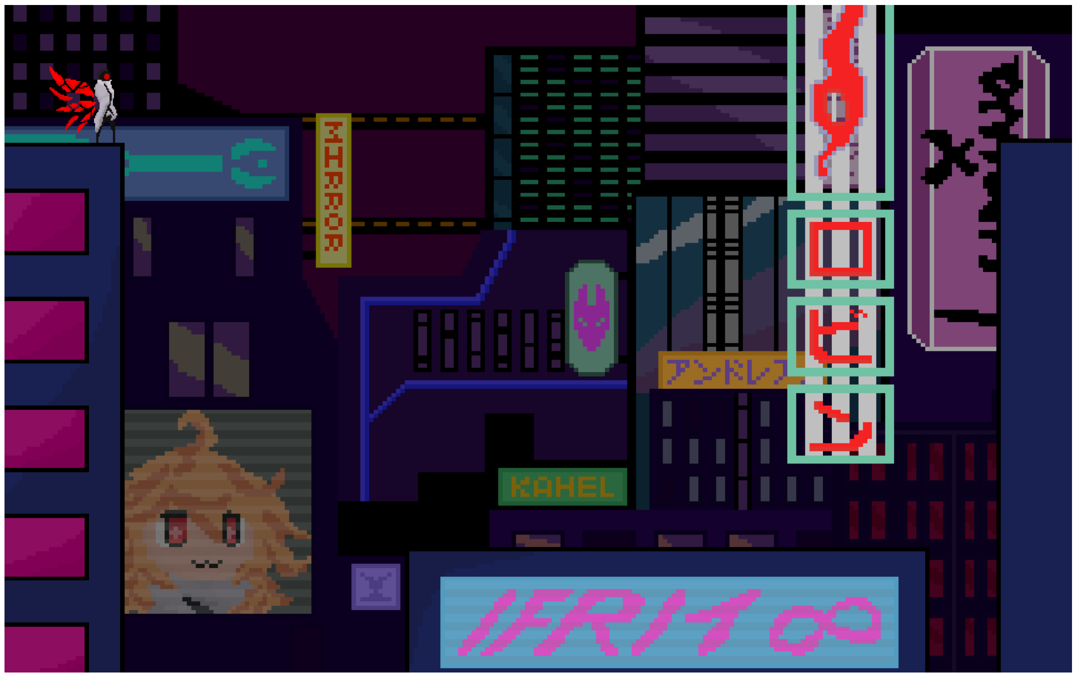

Figure 1. Mockup of a player loading into level 1

3. Project Timeline

Refer to Figure 4.

The first item is where we propose the idea of a 2D platformer game.

The second item sees us writing up use cases and setting requirements. Concepts are put into place with mockups being created.

The third item is where the project is updated with designs that are more fleshed out and close to final. Initial structure ideas are outlined in a preliminary UML.

The fourth item has movement being more or less completed, with wall and ground collision now being worked on.

The fifth item has collision being finished or nearly finished. Level loading has begun implementation or is approaching implementation.

The sixth item has level loading finished or nearing completion. Menus have begun implementation. Bug testing has begun.

The last item has the project being completed, having met all goals and running with few bugs.

4. Project Structure

From main, the menus will be loaded. From here, StartMenu will contain methods for all of the choices, which will start the game, access the LevelSelect, or close the program. The LevelSelect and StartMenu can both load into the gameplay. Game runs the gameplay. From here, collision is loaded for the stage from a Collision object, which is a child of the Load class, which handles the loading of the correct stage. The Character is also loaded, which contains all of the information of the character, including the character's position and collision. The movement classes helps with Character movement handling, such as jumping, horizontal movement, and jumping collision.

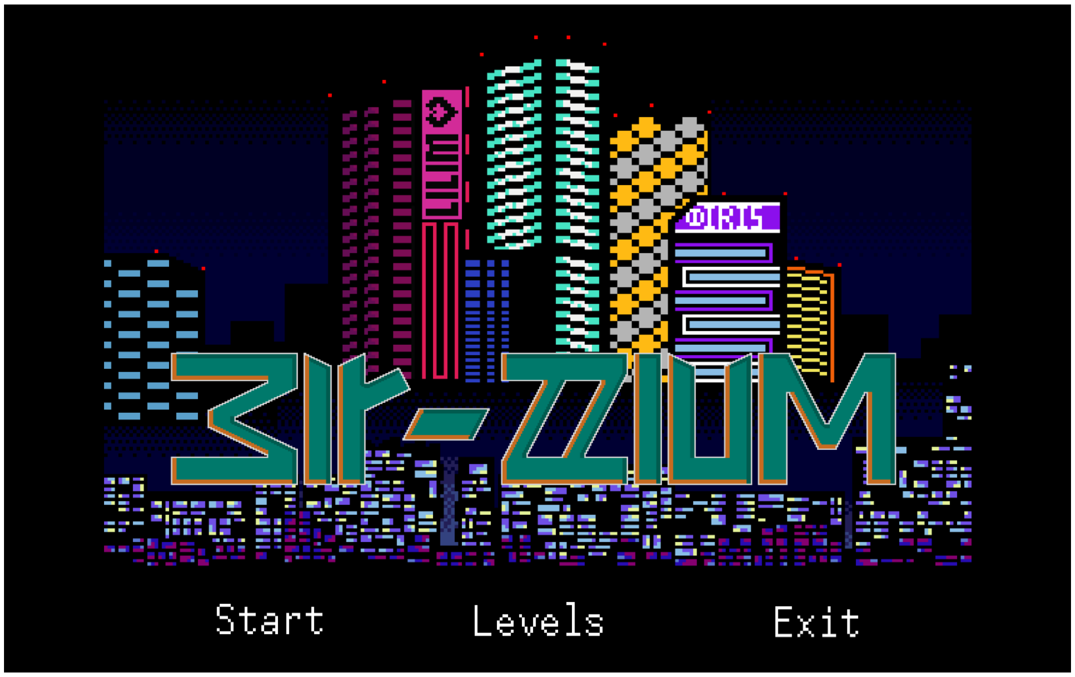

Figure 2. Mockup of the main menu screen

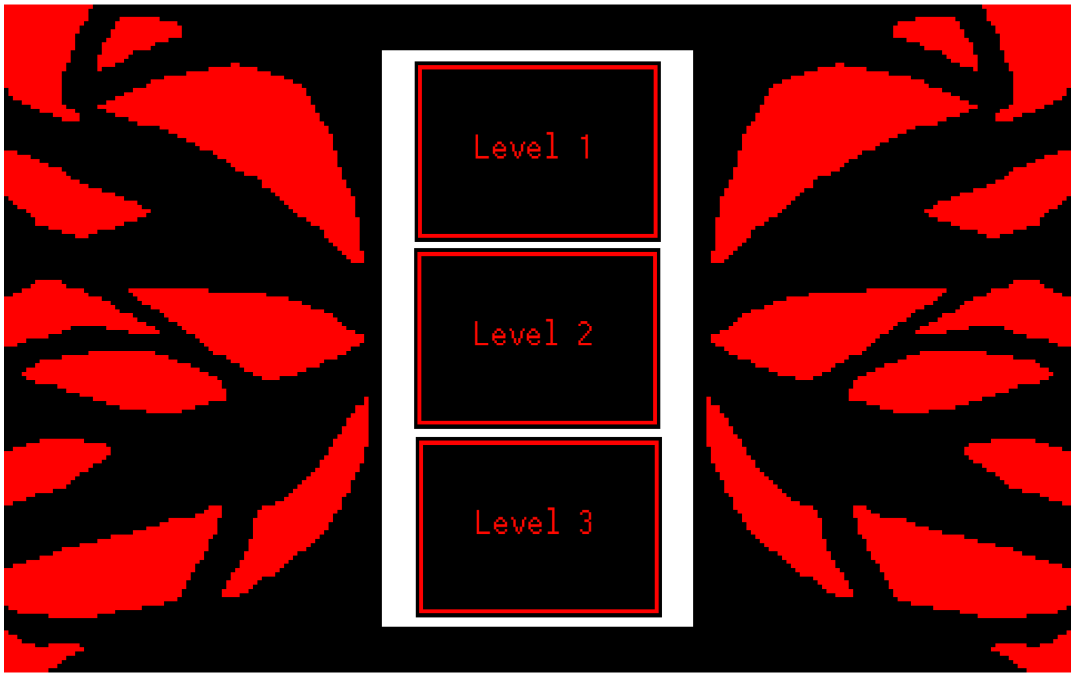

Figure 3. Mockup of the level select screen

4.1. UML Outline

Refer to Figure 5.

Title is what the user loads into upon start up. LoadLevelSelect() creates an instance of LevelSelect, and StartGame loads an instance of Level1. CloseGame() simply closes the program.

LevelSelect is where the user can select any of the 3 levels. LoadLevel1() creates an instance of Level1, LoadLevel2() creates an instance of Level2, LoadLevel3() creates an instance of Level3, and GoBack() goes back to Title.

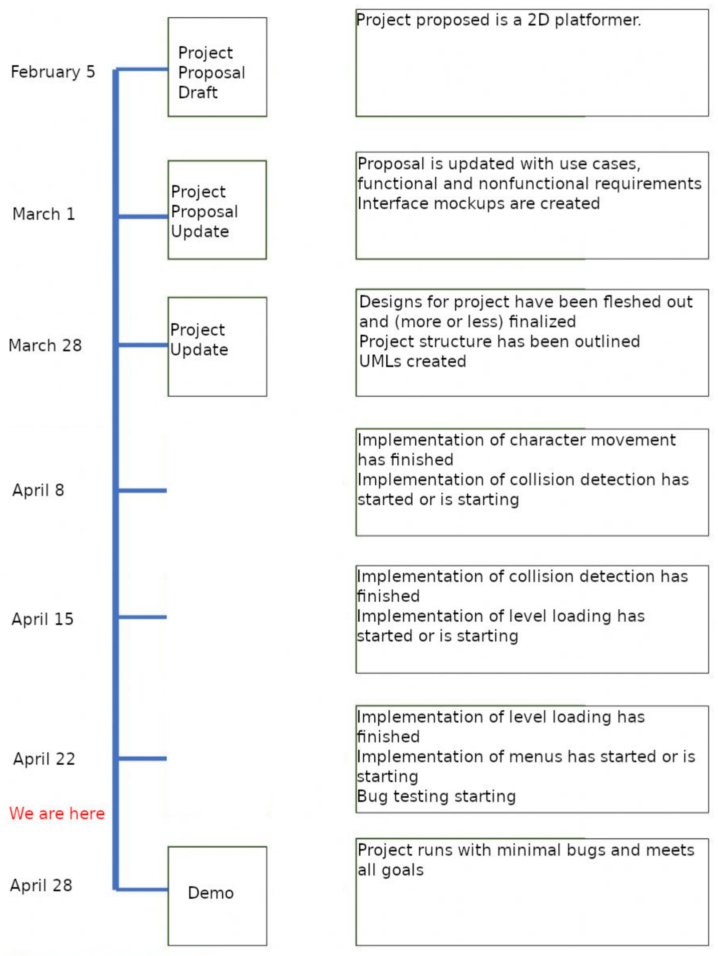
February 5
Project Proposal Draft
Project proposed is a 2D platformer.
March 1
Project Proposal Update
Proposal is updated with use cases, functional and nonfunctional requirements
Interface mockups are created
March 28
Project Update
Designs for project have been fleshed out and (more or less) finalized
Project structure has been outlined
UMLs created
April 8
Implementation of character movement has finished
Implementation of collision detection has started or is starting
April 15
Implementation of collision detection has finished
Implementation of level loading has started or is starting
April 22
Implementation of level loading has finished
Implementation of menus has started or is starting
Bug testing starting
We are here
April 28
Demo
Project runs with minimal bugs and meets all goals

Figure 4. Timeline

Each of the Level child classes contains a Player object (the object the user controls), a List of Surface objects (the walls/grounds in the stage), and a Goal object. These vary depending on the level. The Player object inherits from the RectangleAdapter. It contains a Rectangle, is DispatcherTimer, and booleans for player movement. The object is created with x and y coordinates, which determine where the object spawns on the screen. The keyisdown and keyisup function handles movement when pressing down and releasing the arrow keys or action keys. The dtClockTime_Tick function refreshes the object's state every few frames.

The Surface object and Goal object are both children of the Collision interface and also inherit from RectangleAdapter. Both contain a hitbox Rectangle, a Player object, and contain a DispatchTimer. Both take a Rectangle and Player object as arguments. The Goal object contains a bool that represents if it is or isn't making contact with the Player object. dtClockTime_Tick in the Surface class handles collision between it and the player every timer tick. The Goal class checks to see if the Player has made contact with the object. If it has, the bool is set to true.

The RectangleAdapter class adapts a Rectangle into a RectangleAdapter type, which allows Surface, Goal, and Player objects to inherit from it. This is to get around the Rectangle class being a sealed class.

4.2. Design Patterns Used

The 2 design patterns used are the Strategy pattern and the Adapter pattern.

Strategy: On the level select, a different level is chosen depending on what button the user selects on that screen. Depending on the selection, either of 3 strategies (Levels) can be instantiated: Level1, Level2, or Level3.

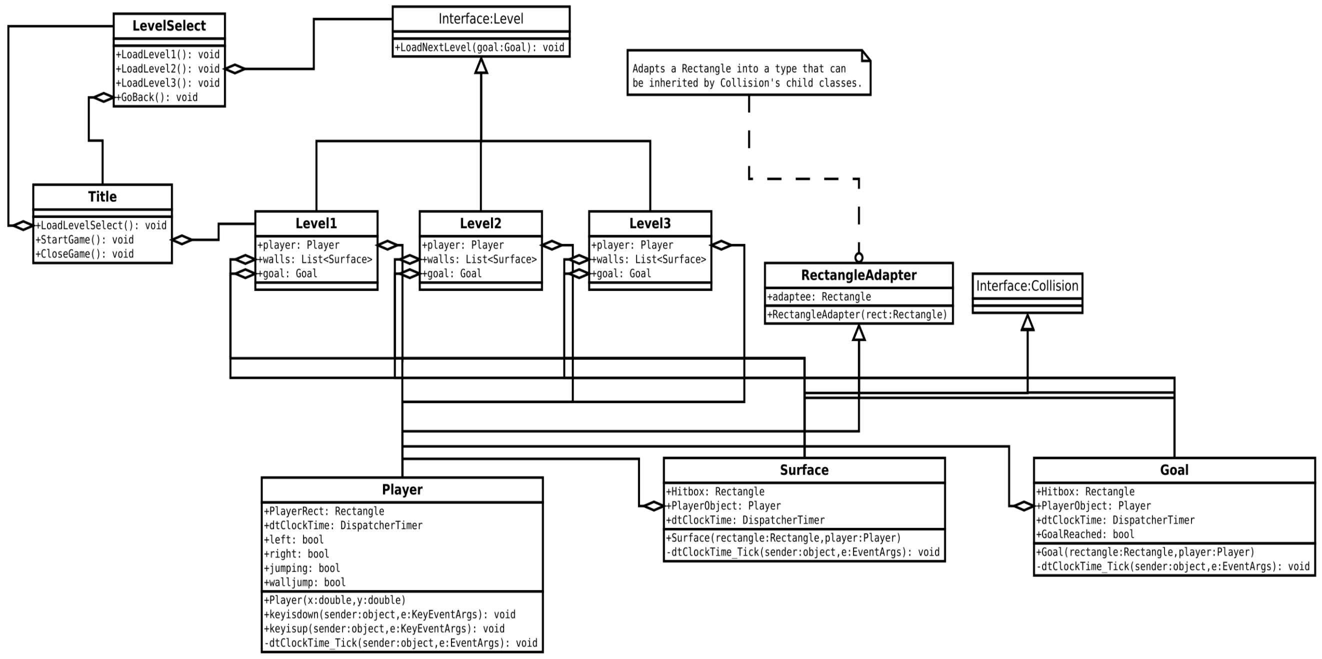
LevelSelect
+LoadLevel1(): void
+LoadLevel2(): void
+LoadLevel3(): void
+GoBack(): void
Interface:Level
+LoadNextLevel(goal:Goal): void
Adapts a Rectangle into a type that can be inherited by Collision's child classes.
Title
+LoadLevelSelect(): void
+StartGame(): void
+CloseGame(): void
Level1
+player: Player
+walls: List<Surface>
+goal: Goal
Level2
+player: Player
+walls: List<Surface>
+goal: Goal
Level3
+player: Player
+walls: List<Surface>
+goal: Goal
RectangleAdapter
+adaptee: Rectangle
+RectangleAdapter(rect:Rectangle)
Interface:Collision
Player
+PlayerRect: Rectangle
+dtClockTime: DispatcherTimer
+left: bool
+right: bool
+jumping: bool
+walljump: bool
+Player(x:double,y:double)
+keyisdown(sender:object,e:KeyEventArgs): void
+keyisup(sender:object,e:KeyEventArgs): void
-dtClockTime_Tick(sender:object,e:EventArgs): void
Surface
+Hitbox: Rectangle
+PlayerObject: Player
+dtClockTime: DispatcherTimer
+Surface(rectangle:Rectangle,player:Player)
-dtClockTime_Tick(sender:object,e:EventArgs): void
Goal
+Hitbox: Rectangle
+PlayerObject: Player
+dtClockTime: DispatcherTimer
+GoalReached: bool
+Goal(rectangle:Rectangle,player:Player)
-dtClockTime_Tick(sender:object,e:EventArgs): void

Figure 5. UML Outline

Figure 6. Morale Support

Adapter: The adapter pattern is used to adapt a Rectangle into being a RectangleAdapter, which is to get around the fact that Rectangle on its own cannot be inherited from, as it is a sealed class. The Surface, Goal, and Player objects then inherit from RectangleAdapter.

5. Results

The project was definitely going to be hard to implement from the very moment we made the initial concept. Making a platformer in a WPF application is certainly not easy or ideal. Regardless, I'm satisfied that we managed to get something together that meets the requirements we established. The project is nearing completion, and it seems we'll have everything we wanted to include at minimum. Looking back, perhaps it wasn't the best idea to create a platformer with a WPF application.

5.1. Future Work

From here, we plan to potentially clean up what we have even further. After all is said and done with this project, we don't have any particular plans to do anything more with this project when it comes to distribution or anything else. We figure this will just remain as a past project.

References

[1] Wasn't sure how to remove this.