”Phantastic Wizard”

Project Description

Lars Erik Faber, Studentnr. 193173

Contents

1 Requirements To Run Application

The only requirements to run the application is to open the solution folder and run in debug mode by hitting run project or F5.

2 Unfinished Parts

2.1 Game Files

The cactus enemy

For a while I had been working on another enemy type to mix up the gameplay. The idea for this enemy is that it is a purple cactus that will target a point in the arena room and grow outwards in four directions, separating the room into four areas. The player would then be able to destroy each of the four stems by damagin it with spells. The code is still left in the final product, but I only got around to drawing the spritsheet for it and coding basic behavior.

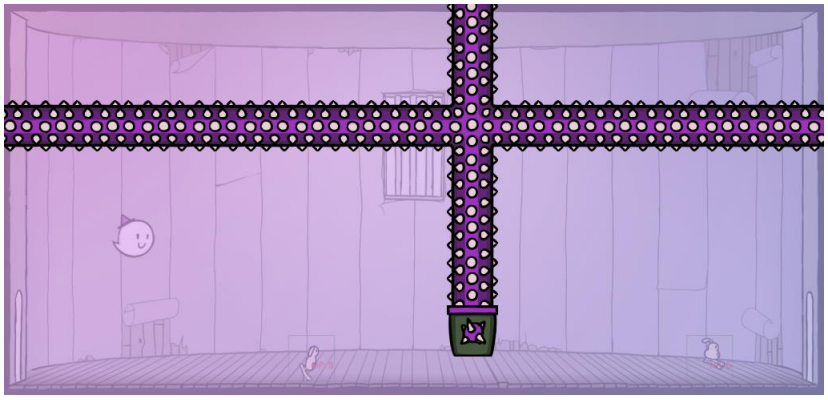

Figure 1: Concept for the purple cactus enemy

2.2 Application pages

The settings page

Originally, the plan was to let players adjust application settings and store them as settings configurations on the database. However, it seemd to overwhelming and unecessary to connect all the settings details to the game, such as adjusting display mode, audio and keyboard button mapping. The configuration model class is still in the model and the database, along with the settings page, but they are not fully implemented. I also realized I could have utilized the settings page from Windows Template Studio, but at that point I had moved on.

The spell book

Originally, the intent was that the player would have to memories combinations of button presses to fire spells. For example, the ice-spell would have a combo of up-up-down-down-up using the arrow keys, but I did not get to code it. This xaml page would give a listing of all the different button combinations required to initiate each spell. However, now it just rests in the main menu and serves no purpose.

3 Known Issues

3.1 Minor Issues

- When creating a new player profile, the xaml will display wrong id in the seleced tab to the right.
- After reaching wave 10 in the game, the EnemySpawner classes get confused, and increments the wave incorrectly.

3.2 Major Issues

- Currently, the leaderboards is not fetching the data correctly, resulting in a strange listview result.

4 App Description

4.1 General Information

About the game

WizardGame is a small video game where players play as a tiny wizard ghost who can cast different spells. The objective is to survive as many waves of enemies as possible without dying, however this becomes more difficult as the waves progress. The enemy count and their damage output increases with the current wave, making it tougher and tougher to survive.

Another important place is the leaderboards. This is where players compete to get thee highest wave, crowning them as number one on the board. The list is ordered by each players personal best game in descending order. Players can also view their previous game to monitor their improvement.

For this to be possible, each place have the ability to create their own player profile that store previous games, and will be displayed on the leaderboards. If they are unhappy with their decisions, they can update their player profile to give a more soothing name, or delete it alltogether.

Assets And Graphics

All the textures and sprites for the game (items, characters, backgrounds, logo, etc) are drawn by me using digital editing software.

4.2 Functionality

Gameplay

To move the ghost around use the W, A, S and D keys, and to cast spells use the up-, left-, down- and right-arrow keys. A wave progresses if all given enemies are eliminated. A normal wave consists of bunny enemies, and a special wave consists of card enemies and will occurr every ten waves. If a bunny or card enemy touches the ghost, the ghost will lose some health points. Once the health points reach zero, the game is over and it proceeds to save the game to the selected profile.

The arrow keys will cast different spells that are used to damage and evade the enemies. The most useful spell is the ice-spell (up-arrow) that shoots horizontaly in the headed direction and will deal a lot of damage to the enemies.

Application Pages

On launching the application, the user is greated with a titlescreen displaying the name. Press any button left-click to proceed to the main menu. From the main menu, it's immediately possible to navigate to the settings screen, player screen, and spell book screen. Only once the database connection has loaded in the viewmodel, do the leaderboards and game become available. During the main menu it is possible to play around with the ghost character.

In the player page, users can select, create, update or delete any desired player profile. In the left panel, all existing players are displayed and ready to be handled by clicking on them. The selected player profile is each game will be tied to when playing. If no profile has already been selected, the rest api did not find a selected player, the application will instead select a default profile.

The leaderboards page will list all players and their best game (highest wave) in descending order. This is also the page that lists a user specific games history, and can be found by using the pivot headers. The page only serves as a monitoring tool to track ones individual progress.

The settings page and spell book page are not fully implemented.

API communication

Fetching data from the database begins in each of the ViewModels. These provide a range of useful methods that fetches and manipulates data in certain ways. For example, the LoadSelectedPlayerAsync method in PlayerViewModel makes a call to api/Players/Selected, which is a custom API route, to fetch the newly selected player. It implements a simple form of error handling to make sure the SelectedPlayer property will a Player object.

When making calls through the Restful API, the application uses the HttpDataService provided by Windows Template Studio. It can be found in the WizardGame.App.Core.Services namespace.

Game Logic

The actual game files occupy a large portion of the UWP, and includes some useful manager classes. The EntityManager class is solely responsible for handling all game entities in the world. That is, anything that can be drawn on screen. It encorparates a layer system that sorts the list of game entities based on its desired layer. This way ensures that the entities are drawn in correct order. It also includes many helper methods to manipulate entities and their properties. They are used everywhere in the code to provide the math and simulate the game world.

The GameStateManager keeps track of the current wave, enemies defeated and time elapsed. It also manages the next wave and determines its outcome. The codebehind for the game page interacts with this class to retrieve the correct data when saving a new game statistic to the player.

Lastly, the AnimatedCanvasControl provides three essential events that provide the basis for the whole game. These events are the ResourceCreate-, Update- and Draw-events. All resources such as loading images and instantiating basic game setup is done here. Next, the update event makes sure to update the state for each entity, such as movement and current actions. Finally, the draw event calls the draw method in each entity to draw them on screen. Most entities also implement the IDrawable interface, that provide the Update and Draw methods.

Other useful classes are the StyleSheet, and ImageLoader classes, that help load images and draw specific parts of the bitmaps to the screen, and the KeyBoard and InputKey classes that handle the keyboard inputs. I have also create some classes and moved them to the Helpers folder (CanvasDebugger, ControlHandler and RandomProvider). The rest of the game related files are located inside the GameFiles folder.

4.3 Screenshots / Concepts

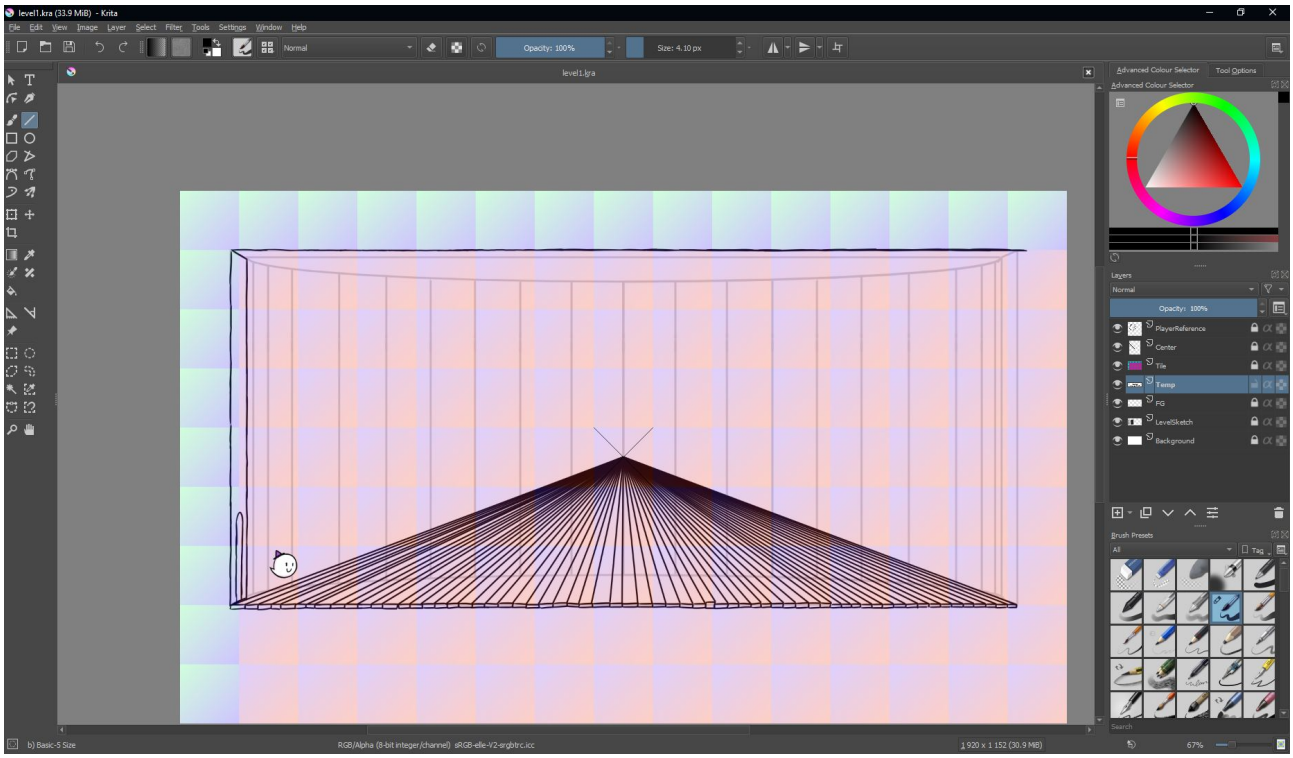

Figure 2: Early draft on the level background

Figure 3: Late development, displaying collision bounding-boxes

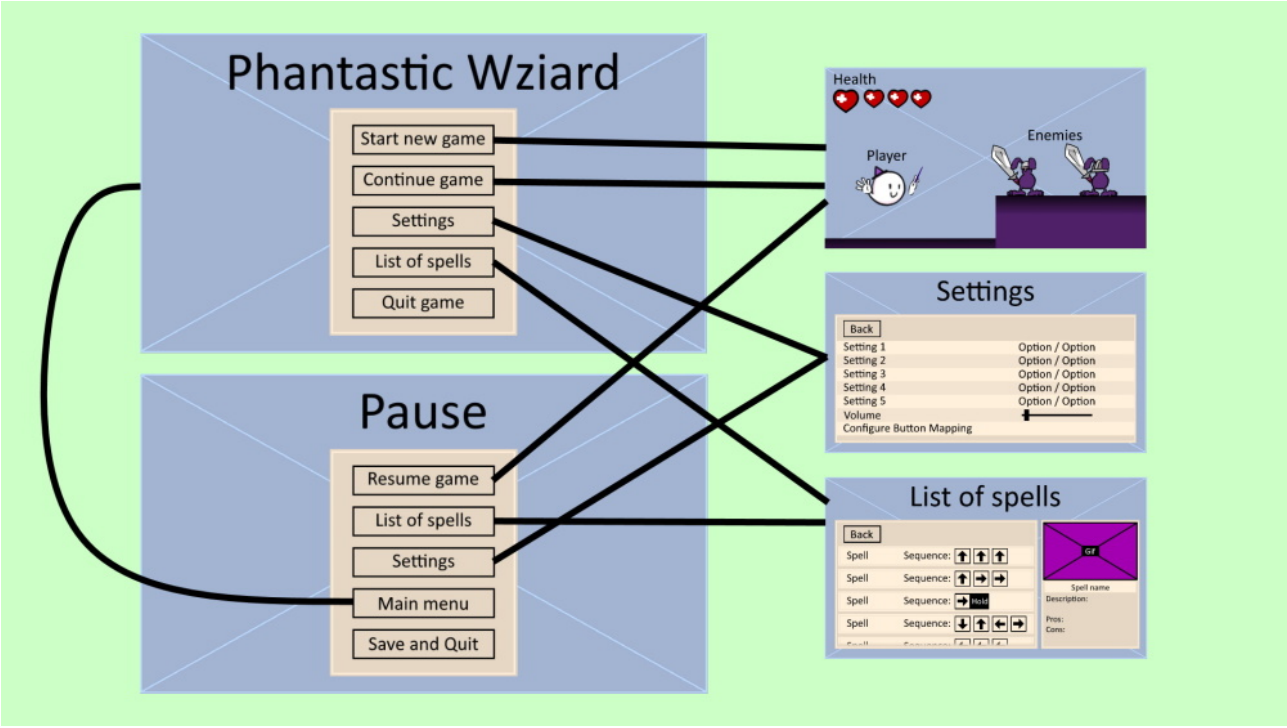

Figure 4: Menu structure sketch

Figure 5: First iteration of main menu