TILE MATCH REPORT

[By Jamaine Christian Jac105]

DECEMBER 6, 2023

Abstract

The goal of this project is to design and develop a tile-matching puzzle game using unity, making use of the Tilemap feature of the engine to create levels and gameplay elements. The game has 2 levels of differing difficulty, an intuitive UI for fluid player interaction, and a variety of unique power-ups that provide strategic depth to the gameplay. Game states, tile management, and the implementation of special tile effects like bombs and colour matching power-ups are all handled by the game controller, which controls the main game logic. The Scene Loader is intended to facilitate smooth interaction between the UI and the game such as level transitions. The Grid Manager is used to create an instance, so things can be stored between scene transitions, for easy processing of variables needed in the next scene but also need to retain their value.

Introduction

The assignment was to create a tile matching game in Unity with C# that has unique features like break four in a square, sand mechanics and special power-ups. The game was required to have at least 2 level and an interactive UI. The difficulty in this project was within the sand mechanics and making sure the power ups were working without conflict. I chose to use tilemaps as the matching game would need to be a grid-based game.

Software Design

USER INTERFACE

The user interface was created using TextMesh Pro textboxes, images and and buttons in a way that boosts the user experience due to its simplistic, minimal design. The UI is separated into separate layers using panels, which are set to active or un active depending on button presses or scene switches. The logic behind the transition of the UI is handled within the Scene Loader script, which is intended to switch scenes and update which UI display/menu should be shown on screen.

GAME LOGIC

The game logic is mostly handled within the GameController script. The game controller script initialises the tiles if the selected scene is currently one of the playable levels. The game then runs of an enum class called GameState that is used to handle the state of the game within the update function. The game either loops the logic for playing or is stuck in a loop on a menu until it is broken by a button click.

The Play state is comprised of many functions working in conjunction with each other to produce the playable game. This method checks the game ending conditions such as winning or losing and if either are met then the game state will switch to either victory or game over. It then gets the tile at the position of the mouse clicker and removes the tile if it is a breakable piece. The logic for each power up has its own method but this is all passed into removeTiles which will remove a list of tiles. This removesTiles then starts the sand mechanic so the grid will be filled correctly after removal. If the list is empty it waits for a second click to then try and swap the tiles. The logic to check if they are swappable pieces then runs and swaps them if they are.

The pause state has an integer to store what state of the menu it is in also. It uses this to switch between the rules menu and the pause menu displays.

Conclusion

This project highlighted my strengths and weaknesses. My strengths are the logic behind the game and my weaknesses are using the UI within unity. This Project was a partial success, the majority of functional requirements was working before I changed the way the UI is displayed (onto panels) and now it stops functionality of the pause menu and button selection in some scenes. All powerups work except concretion. Sand is also a working mechanic.