HEX 2F - Team 47

Project Title: JUNK

https://github.com/fernandezmatthew/Hex2FProject

Matthew Fernandez, Yifan Zhu, Phillip Vanderlaat,
Anthony Long

Table of Contents

Project Description

- Project Description
    - We built a local multiplayer, co-op game. The objective of the game is to collect all the trash in each level in the quickest amount of time. There are two players, each having a unique game experience, who need to work together to solve the puzzles that obstruct them from collecting all the trash.

    - Our game is for young people who may not be aware of the growing problem of climate change and it will improve their awareness on this topic.

- How solution addressed challenged statement
    - Our game is meant for young people who may not be aware of the growing problem of climate change, our project is a co-op game that improves awareness of the importance of environmental conservation through the use of theming. Our co-op game will be focused around picking up trash and solving puzzles that incorporate environmental concerns.

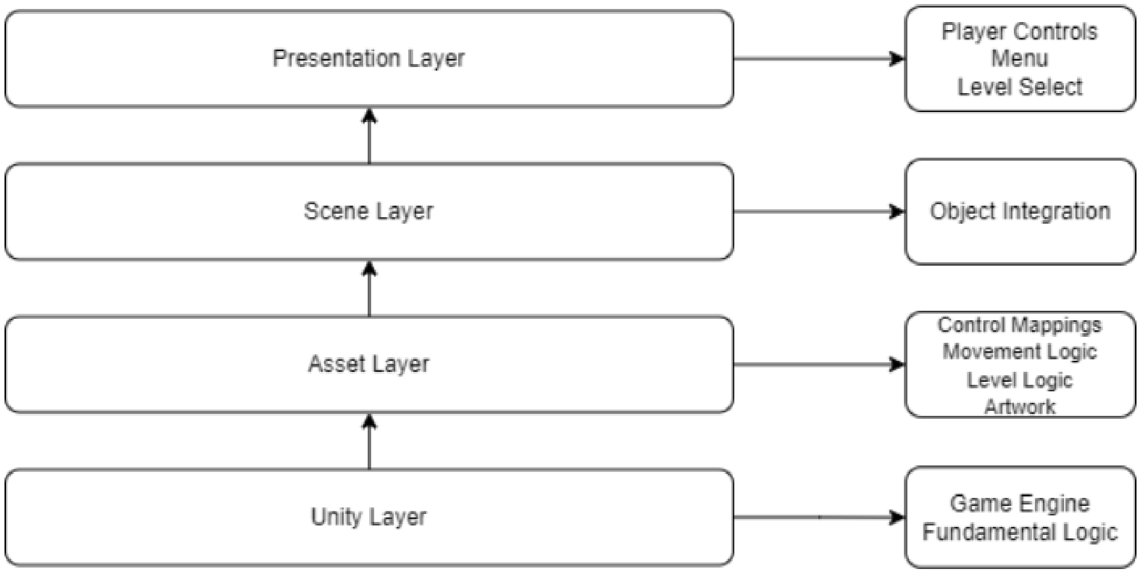
- Features and Functionality
    - There are two character types, a land and a water character:
        - The land player moves and jumps fast on land, but it cannot swim underwater the water (it is always forced to stay afloat and swim on the top of the water).
        - The water player swims fast in the water and can jump high in the air from a submerged state. If it is on land, it hobbles around and cannot jump very high.
    - Each level contains puzzles such as obstacles / hazards, manipulatable triggers, and a unique game environment for both characters.
    - Users must sign in to record their state information
    - Level high scores are recorded in a database and can be improved upon later.

System Models:

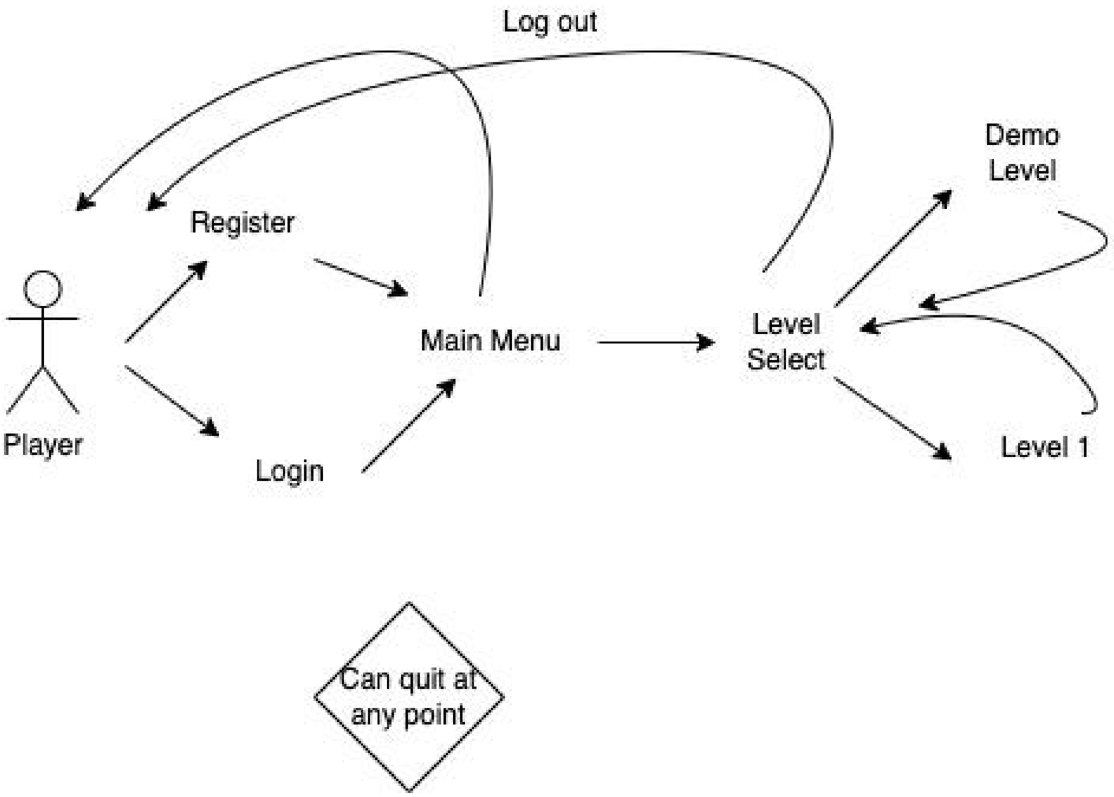
The Architectural pattern

- Our game is a monolithic application with component-based functionality
    - Not Microservices
    - Not Client-Server
    - Local two player co-op system
    - Layered
    - We will be developing the Presentation and Functionality Layers
    - Unity handles all the backend layering

Layered System Context Model

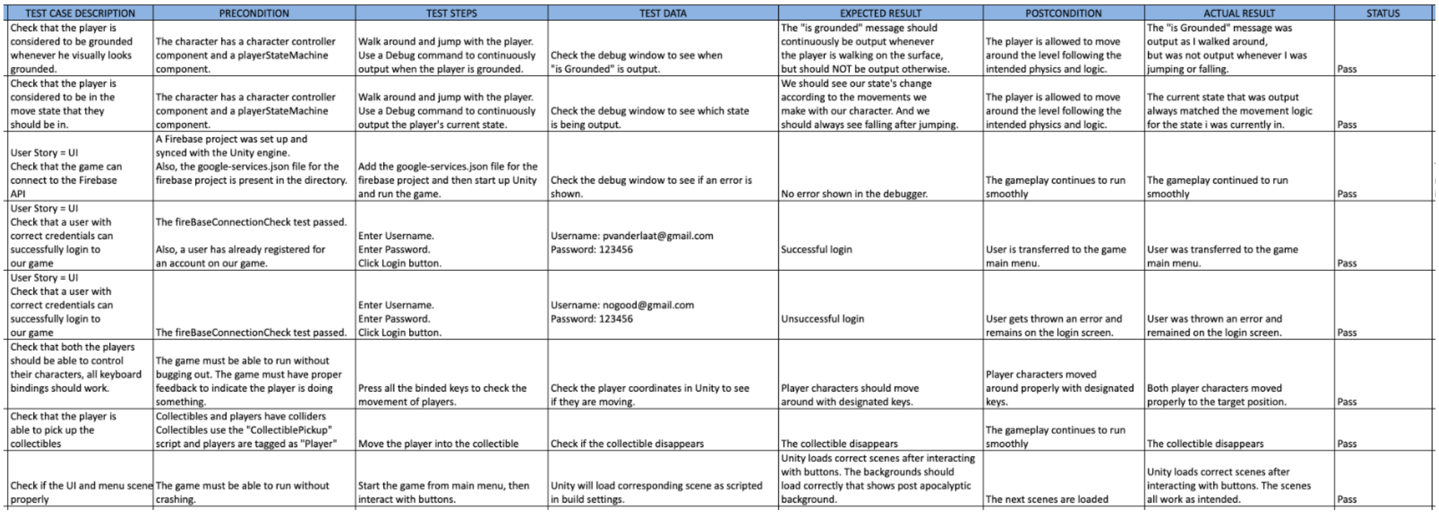


Use Case Model



Code Management

- Code Management
    - Our project, including our code, is remotely stored using Github. We did all of our coding using Unity's built-in integration with Visual Studio. These gave us access to an intellisense system capable of understanding Unity's many functions.
- Test Plan
    - Our plan for testing involved building functionality and testing as we go.



TEST CASE DESCRIPTION	PRECONDITION	TEST STEPS	TEST DATA	EXPECTED RESULT	POSTCONDITION	ACTUAL RESULT	STATUS
Check that the player is considered to be grounded whenever he visually looks grounded.	The character has a character controller component and a playerStateMachine component.	Walk around and jump with the player. Use a Debug command to continuously output when the player is grounded.	Check the debug window to see when "is Grounded" is output.	The "is grounded" message should continuously be output whenever the player is walking on the surface, but should NOT be output otherwise.	The player is allowed to move around the level following the intended physics and logic.	The "is Grounded" message was output as I walked around, but was not output whenever I was jumping or falling.	Pass
Check that the player is considered to be in the move state that they should be in.	The character has a character controller component and a playerStateMachine component.	Walk around and jump with the player. Use a Debug command to continuously output the player's current state.	Check the debug window to see which state is being output.	We should see our state's change according to the movements we make with our character. And we should always see falling after jumping.	The player is allowed to move around the level following the intended physics and logic.	The current state that was output always matched the movement logic for the state i was currently in.	Pass
User Story = UI Check that the game can connect to the Firebase API	A Firebase project was set up and synced with the Unity engine. Also, the google-services.json file for the firebase project is present in the directory.	Add the google-services.json file for the firebase project and then start up Unity and run the game.	Check the debug window to see if an error is shown.	No error shown in the debugger.	The gameplay continues to run smoothly	The gameplay continued to run smoothly	Pass
User Story = UI Check that a user with correct credentials can successfully login to our game	The fireBaseConnectionCheck test passed. Also, a user has already registered for an account on our game.	Enter Username. Enter Password. Click Login button.	Username: pvanderlaat@gmail.com Password: 123456	Successful login	User is transferred to the game main menu.	User was transferred to the game main menu.	Pass
User Story = UI Check that a user with correct credentials can successfully login to our game	The fireBaseConnectionCheck test passed.	Enter Username. Enter Password. Click Login button.	Username: nogood@gmail.com Password: 123456	Unsuccessful login	User gets thrown an error and remains on the login screen.	User was thrown an error and remained on the login screen.	Pass
Check that both the players should be able to control their characters, all keyboard bindings should work.	The game must be able to run without bugging out. The game must have proper feedback to indicate the player is doing something.	Press all the binded keys to check the movement of players.	Check the player coordinates in Unity to see if they are moving.	Player characters should move around with designated keys.	Player characters moved around properly with designated keys.	Both player characters moved properly to the target position.	Pass
Check that the player is able to pick up the collectibles	Collectibles and players have colliders Collectibles use the "CollectiblePickup" script and players are tagged as "Player"	Move the player into the collectible	Check if the collectible disappears	The collectible disappears	The gameplay continues to run smoothly	The collectible disappears	Pass
Check if the UI and menu scene properly	The game must be able to run without crashing.	Start the game from main menu, then interact with buttons.	Unity will load corresponding scene as scripted in build settings.	Unity loads correct scenes after interacting with buttons. The backgrounds should load correctly that shows post apocalyptic background.	The next scenes are loaded	Unity loads correct scenes after interacting with buttons. The scenes all work as intended.	Pass

- We manually tested many things along the way using Unity's logging system. Below is a list of tests that we created along the way and used to keep our project running smoothly.

- Static Code Analysis and Report
    - Unfortunately, Unity projects are not well suited to the popular static code analysis tools currently available. We were unable to perform a static code analysis + report for our project.

Technical Details

- Installation Instructions
    - Developer installation instructions
        1. Clone the repository
        2. Download the Unity-Firebase SDK and import the package as a “custom asset”
        3. Import the google-services.json key from the firebase dashboard
    - User installation instructions
        1. Only works on windows platform
        2. Download the .exe and play!

- Login and Access Credentials
    - Users must log or register upon starting the game
    - The login and database functionality is implemented using FireBase

- API Keys
    - The login and database functionality is implemented using FireBase which utilizes a json key to authenticate our program
        - In order for our game to communicate with our database, we registered our game with the firebase project and ensure the firebase project’s JSON key is present in the game’s project directory

Risk Management

- Risk Management Plan
    - Unity handles OS resources, prevents crashes, and developer logs
    - Safe distribution to prevent impersonation by hackers
    - Project management Feasible goals and timeline

- Software Quality Attributes
    - **Availability**: Because our game is quite simple, it is able to achieve the intended purpose with no failure. There are no game breaking bugs.

    - **Performance**: Our game loads quickly and is able to communicate with the database seamlessly. Our game's performance is very strong.

    - **Testability**: Video games can be hard to test (aside from manual testing). We were only able to perform integration and end-to-end tests through manually playing the game.

    - **Security**: All security aspects involved in our game are in the hands of either Unity or Google.
        - All resource management is handled by Unity, so we as developers have little control over this aspect of our game's security
        - All login functionality and user data storage is handled by Google's Firebase product, so we have little control over the user's data security in this regard

References

- Our game was inspired by *Fireboy & Watergirl: Elements*, a co-op online game.
    - https://www.coolmathgames.com/0-fireboy-and-water-girl-in-the-forest-temple
- Presentation icons by flaticon.com