Blob the Adventurer,

A game by:

John Metcalfe - 2616981

Brandon Caudell - 2650354

Adam Heald - 2672863

Goals

The goals of this project were to:

* Make a simple platforming game
* Adapt the general software development techniques presented in the course to game design
* Make something that would be different than most projects

Updated Project Proposal

For our semester project we would like to make a two dimensional side-scrolling game. The character will be controllable by the player and will be able to move throughout a 2D environment based on the user's input. The character will be able to move both horizontally and vertically across the screen and will react to the environment with an at least a simplified physics interaction. The player will have to move mainly horizontally through the level to reach the end while avoiding or defeating enemies and avoiding falling off platforms. There will be at least one level, possibly more if time permits. If there is more than one level then after one level is completed the next will load and the player's score will persist.

Steps:

1. Create class, activity, and state diagrams for the relevant systems (i.e. menus, AI, general game objects)

2. Plan an MVC architecture to handle player input and update the game's underlying data.

3. Use methods from Agile Development to implement basic functionality of the game to a

playable state. Then add additional functionality incrementally.

4. With left over time we will adjust already implemented features based on feedback from

playing the game (not necessarily bugs but making the game play well).

5. Finally we will try to make the game look good by adding art if there is time.

The output of the program will be the visual display that the player sees as they are playing the game. The game will provide visual feedback to the player both in the form of GUI and visual cues with the in-game 2D graphics.

GUI

The GUI in our game was kept to a minimum to avoid overloading the player’s vision. We tried to convey all the essential information in as little GUI as possible. Our main focuses were to make sure the user could easily navigate the levels and see how much health they had left as well as the score they had accumulated. The main menu GUI consists of 5 buttons; one for each level and a fifth to start the game from the beginning. The gameplay GUI consists of multiple hearts representing the amount of health the player has left in the top left corner and a number representing the player’s score in the top right corner.

Originally it was tough to tell when the player was getting damaged unless the player was looking at the hearts constantly so to make it more obvious we made the player change colors for a short amount of time. This made it easier to let the player know they’ve been hurt since they are most often looking at the player.

Another problem was letting the player know how to move. The simple way would be to put it on the gameplay GUI however we did not want to clutter the main GUI. The solution we found was to put the information in the game level as floating text that moved with the world. This let us also put the information in a context where the player would need to know it.

Lastly, we made sure that each of the 2D sprites representing the characters and environment in the level matched their 2D physics representations so that the player would not experience discrepancies between what they see and what is actually happening.

Tasks

* Create the player
* Create platforms
* Create traps
* Create basic enemy
* Create test level
* Make camera follow player
* Create level transition system
* Create Main GUI
* Create level 1
* Create level 2
* Create level 3
* Create level 4
* Create Victory Level
* Update enemy behavior
* Add handling for player leaving level bounds
* Add player respawning
* Create game art
* Create level selection
* Add music – Music written by Max Moore

Testing Process

Due to the amount of variability in positioning and user input in video games we were not able to discretely test every possible input since the amount of inputs are infinite.

* + After the player was implemented each button corresponding to the player’s input was pressed to make sure the player’s position changed according to what was expected
  + After the platform was implemented the player was moved above them to make sure the player did not fall through
  + After traps were implemented the player was moved on top of them to make sure 1 point of health was removed from the player
  + After the basic enemy was implemented it was placed over a platform to make sure it turned around at the edges of platforms
  + After each of the above was implemented a test level was created which implemented each of the above elements in one environment for integration testing.
  + After the camera following logic was implemented we move the player in each direction to make sure the camera would follow the player correctly.
  + After the level transition system was completed we created two test levels and made sure the system correctly transitioned between the levels when the player reached the end of a level.
  + After creating the main GUI we tested it by pressing each of the buttons and making sure the level transition system transitioned to the correct level.
  + After creating level one we tested it by having each one of us play through the level multiple times to make sure everything was working as expected.
  + We repeated the same testing as above for levels 2-4.
  + After we finished the Victory level we tested each button on the GUI that appears in the level to make sure it either correctly quit the game or went back to the main menu.
  + After updating the enemy behavior we tested it by placing it between two walls to make sure it moved between them correctly and we again placed it on a platform to make sure it went between the edges correctly still.
  + After creating a system to handle the player leaving level bounds we tested it by purposely trying to escape the levels that we had created to ensure the player could not leave the intended extents of the level.
  + After adding player respawning we tested it by intentionally killing the player in different ways and pressing the respawn button to make sure the level was correctly started over in each case.
  + After creating in game art we tested it by placing it on all the objects it corresponded to and made sure it matched the physics representations bounds in the game world on each piece of art.
  + After creating level selection we made sure each of the buttons loaded the correct level when pressed.
  + After adding music we made sure that it played consistently through the game without turning off or player more than once.

Traceability Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time Table for Tasks and Milestones | | | | |
| Major Tasks | Days | | | Person Responsible |
| April 15 - April 30 | May 1 - May 5 | May 6 - May 10 |
| Create the player |  |  |  | JM |
| Create platforms |  |  |  | JM |
| Create traps |  |  |  | JM |
| Create basic enemy |  |  |  | JM |
| Create test level |  |  |  | JM |
| Make camera follow player |  |  |  | JM |
| Create level transition system |  |  |  | JM |
| Create Main GUI |  |  |  | JM, BC, AH |
| Create level 1 |  |  |  | JM |
| Create level 2 |  |  |  | BC |
| Create level 3 |  |  |  | AH |
| Create level 4 |  |  |  | JM |
| Create Victory Level |  |  |  | JM |
| Update enemy behavior |  |  |  | BC |
| Add handling for player leaving level bounds |  |  |  | JM |
| Add player respawning |  |  |  | BC, AH |
| Create game art |  |  |  | JM, BC |
| Create level selection |  |  |  | AH |
| Add music |  |  |  | JM |
| Iterative Development | Intial Playable Build | Playable with Levels | Full Game | JM, BC, AH |
|  |  |  |  |  |
| Initial Work |  |  |  |  |
| Updating / Improvement |  |  |  |  |

Discussion

The majority of the primary functionality was handled behind-the-scenes by the Unity engine. Because of this, the performance of the application is very good. The engine is very efficient in executing our project, and we did not observe any noticeable lag in the game. Details such as sprite collision are handled by the engine with relatively-high accuracy. The entire game is presented in a very user-friendly fashion, with simple controls and UI. Any user that has played a 2D platformer such as this before will have a very easy time using the application.

GitHub

Project Repo: <https://github.com/JMetcalfe201/448_SemesterProject>

We utilized GitHub as our version control system. Since the three of us were not developing at the same time and in the same location very often, this allowed us to have the latest version of each other's code. Thanks to prior experience in labs and homeworks with GitHub and our chosen IDE's of choice, we had very few hiccups with the system. It made developing this project very easy.

Future Extentions

Due to time constraints, there was a lot of content that we were unable to include in this game. At the time of this report, future content might include additional levels, enemy types, power-ups for the player, bosses, and a high-score board. The majority of this material falls into the category of “content”, however, and would be rather easy to integrate into the existing game. Since the main components of the game are already in place, the only requirements for these additions would be asset creation, AI development for new enemy types, additional controls for power-ups, and other such details. We hope to implement at least some of this before the demonstration, but at the time of this report, these features are not yet included in the game.

References

* <http://unity3d.com/5>
  + Homepage of the Unity engine.
* <http://docs.unity3d.com/Manual/index.html>
  + Unity documentation.

Software Documentation (scripts)

* AutoAddUIToCanvas.cs
  + Automatically adds the in-game UI to the canvas to be displayed during runtime.
    - Score Text.
    - Hearts.
* AutoRouteThroughMainMenu.cs
  + Displays log error to inform testers how to test correctly when attempting to start the game outside the Main Menu.
  + "When testing you must hit play from the main\_menu level. Then press the number on the keyboard for the index of the level you want to load.\n Make sure the level you want is added in File->Build Options".
* BaseEnemy.cs
  + Movement patterns and death actions.
  + Registers collision boxes and behavior.
  + Incoming damage and health reduction.
* CameraFollow.cs
  + Finds the player and moves the camera along with the player according to the follow speed.
* DebugLevelLoader.cs
  + Allows testers to load to their desired level from the starting menu based on number key pressed.
* DontDestroyMusic.cs
  + Allows music to be persistent throughout game
* EndOfLevelPoint.cs
  + Loads the next level when the player enters the end level collision box.
* KillPlayerVolume.cs
  + “Kill zone” placed below the levels that removes the player’s remaining health when the player enters the collision box.
* LevelManager.cs
  + Adds to the score
  + Times level loading and displays loading screen if necessary
  + Loads levels and destroys main menu buttons on level load
* PlayerController.cs
  + Handles interaction between PlayerView and PlayerModel
* PlayerModel.cs
  + Physics for moving and jumping
  + Handles damage taken and death
  + Collision and grounding functionality
  + Respawns player and reloads current level
* PlayerView.cs
  + Handles user input
  + Updates health display
  + Changes player sprite when player is hurt and back when hurting stops
  + Switches to death sprite on death
  + Displays respawn button on death
* SpikeTrap.cs
  + Deals damage to player if player enters collision box
* TimerManager.cs
  + Contains list of structs that consist of pointers to functions and times for the functions to be called and calls any function whose time has been passed.
  + Increments time based on time since last frame.
* UIGroup.cs
  + Displays and hides UI components