Software Implementation and Testing Document

For

Group - Killer Among Us

Version 2.0

Authors:

Supriya Palli

Thuyvan Vulam

Joseph Ene

Anika Patel

Alec Amico

1. Programming Languages (5 points)

The programming language we used is C#, which is essentially the only option in terms of working with Unity. Additionally, it is a language we are familiar with because it parallels C++ which has been used in previous classes. It also provides us with lots of libraries that make implementation easier, and as it will be used for scripting features and controls, it's important that it is convenient for use with the technology we want to include in our project.

2. Platforms, APIs, Databases, and other technologies used (5 points)

Platform:

- Unity - as a gaming development platform, this is where the entire project was worked on

Plugins:

- Github - used as Github extension within Unity
- Ink - used for integration of Ink within Unity - allows for playing/previewing Ink stories with the "Ink Player" window in Unity

Outside Assets (All inside the "Assets" folder):

- Google Images - used for graphics for backgrounds, static images, etc. - Folders: Sprites & Tiles/Sp_Sprites, Sprites & Tiles/TV_Sprites/Respawn Flags, Animation/AP_Animation
- Glitch the Game - used for getting sprite sheets of characters for our game that could be cut up to be used in the animation of our characters - Folder: Sprites & Tiles
- Kenny Game Assets - used mostly for the platforms in the game - Folders: je_kenney_simplifiedplatformer, Sprites & Tiles/JE_Sprites
- Developed Assets - Andrew Morris developed 3 slime "enemy" assets for us - Folder: Sprites & Tiles/AA_Sprites/Enemies
- Developed Assets - Mark Powell developed spike assets for us - Folder: Sprites & Tiles/AA_Tiles/Functional_Tiles
- Grunge Horror Environment background asset (By ansimuz) - used for the background of the main menu - Folder: JE_Images
- City Pixel Art (By Joseth) - used as a TileSet for the buildings in the platform portion of the game - Folder: Sprites & Tiles/JE_Sprites/Background_elements

- Beside the Fire in the Cavern (By VOiD1 Gaming) - used for the general audio of the main menu - Folder: JE_Audio
- Maid Cleaning Cart (By Aw0) - used for enemy sprite in level two platform scene - Folder: Sprites & Tiles/JE_Sprites

3. Execution-based Functional Testing (10 points)

1. Player presses left, right, or jump button -> Character should move to the left, right, or jump (High)
    a. We tested this function by using the on screen buttons to move the player around the platforms.
2. When the character hits enemy characters, weapons, or falls off platforms -> Character should respawn to beginning of level/back to checkpoint (Medium)
    a. We tested this by making the player collide with an enemy/trigger, then seeing if the character will move back to the beginning of the level or the last checkpoint.
3. Character moves near ghosts -> Ghost dialogue pops up and the player learns more about the story (Low)
    a. We tested this by moving the character close to the ghosts to trigger the dialogue speech bubbles.
4. Player reaches end of platform level by traveling through the door at the end of the level ->Takes player to dialogue portion of the game (High)
    a. We tested this by having the player go through the entire platform level to see if the scene correctly transitions to the respective dialogue scene.
5. Player chooses to start game/choose level selection -> Takes player to the specific game level (Medium)
    a. We tested this by starting at the main menu scene, then choosing the level selection button, then Level Zero. The scene would change into the designated level.
6. Player presses options menu -> Player can adjust the volume (Low)
    a. We tested this by choosing different volume levels and seeing if the volume is affected by this change.
7. Player reaches a checkpoint -> Flag should change colors from red to green (Low)
    a. We tested this by moving the player past the checkpoint to see if the flag changed colors.
8. Player collides with an item -> Item should be “picked up” and disappear from the screen, should be saved into a variable or trigger an action based on object tag (Medium)

    a. We tested this by having the player approach objects with different tags to make sure they were destroyed on collision; and checked if the respective action was being done or the necessary variables were being changed.
9. Player has or does not have item -> triggers or fails to trigger an action based on item possession (Medium)
    a. We tested this by having the player pick up or not pick up items to see the behavior of certain actions.
10. Player presses a pause menu input button -> Player can choose to select resume, restart, main menu, settings, or quit
    a. We tested this by pressing the esc button as a temporary trigger, then the pause menu would pop up. We then tested each of the individual options.
11. Pause Menu Resume button -> Player can resume the game. Unfreezing player and enemy movements, and allow user input to go through again.
    a. We tested this by choosing the “resume” option on the pause menu to resume the part of the game we were at.
12. Pause Menu Restart button -> Level should restart with character being in beginning position for the level.
    a. We tested this by pressing the pause menu and choosing the restart option so that the scene can reload.
13. Player reaches a dialogue scene -> Player should be able to choose options for which dialogue they want to respond with (High)
    a. We tested this by running the dialogue scenes and making sure the buttons were able to be pressed to choose options. Also, we made use of the “Ink Player” window within Unity that was able to show the Ink story and automatically run the dialogue as well.
14. Dialogue scene ends -> Dialogue scene should shift back to level selection screen (High)
    a. We tested this by running through an entire dialogue scene and testing the “Return to Level Selection” button that would appear at the end of the scene.
15. Player reaches falling hazard portion -> player should die if hit by falling hazard (Medium)
    a. We tested this by moving the player towards the hazards and observing if the character would be affected.
16. Player encounters enemy -> player should die if touching enemy (Medium)
    a. We tested this by moving the player towards the enemies and observing if the character would be affected.
17. Player collides with ladder -> player should be able to move up and down ladder (High)

    a. We tested this by moving the player upward when there is a ladder present and seeing if the player would “climb.”

4. Execution-based Non-Functional Testing (10 points)

1. Testing iOS controls -> player should have smooth transition between different functions (i.e. running or jumping)
    a. We tested this by running the scenes and using the on screen controls to see how the animations behaved and how the main player moved
2. Ensure player doesn’t clip into map -> foreground of map should contain composite collider in order to treat the foreground as one continuous object
    a. We tested this by running the scene and seeing if the player would move beyond the boundary of the ground and walls
3. Prevent wall jumps -> ensure player object contains a material which removes friction
    a. Tested by attempting to jump walls with friction, and then adding a no-friction material and retesting
4. Tested the frame rate -> Ran the game in the Unity Editor and checked the FPS stats that the editor monitors on the Game Runtime Tab.
    a. We tested this by playing through a scene while the statistics window was up to give us real time information on frame rate
5. Needs to not take up too much RAM -> Game must be small enough to run on an iOS mobile device
    a. We tested this by running our game on an actual iOS mobile device

5. Non-Execution-based Testing (10 points)

We performed this testing by doing code reviews, inspections, and walkthroughs in our meetings. We would do this by having individual team members screen-share during meetings in order to show their work to the team. Typically, we would start our meetings off by having each member share what work they accomplished in a quick couple of minutes through screen-sharing. This way, all the team members could see the work and be able to offer help or suggestions. For example, a team member may share a script they are having trouble with and walk the group through what the script does in a screen-share, and then the other team members could offer help and advice if needed.