Software

Design

Specification

for

TwinVisitor

Version 1.1

Prepared by:
Jianna Angeles *(Lead Environment Design and Music)*
Vanja Venezia *(Lead Programming and Sound)*
Daniel Arias *(Lead 3D Assets and Story)*
Lucas Wilkerson *(Programming)*

Fall 2022

Table of Contents

1.0 Introduction

1.1 Goals, objectives and Statement of scope

The game will be built using Godot 3D Mono, with 3D assets created in Blender. The core gameplay loop involves switching between two primary characters who can talk to different sets of NPCs to find clues and solve a mystery. Puzzles will be presented to the player in the form of NPC dialogue, interactable items in the game world, and inventory items the player receives. The player will be able to save and load game states to maintain progress between play sessions.

1.2 Software context

TwinVisitor is a 3D low-poly, 3rd-person perspective mystery game, aimed at those who enjoy light puzzles and conversation driven story games. Presently, the game is envisioned as a self-contained proof-of-concept, with the potential for a larger game experience eventually once the MVP is finished. Inspiration has been drawn from many sources and amalgamated into a unique product that stands on its own. If a state of reasonable completion is reached, the game may receive a free release as a demo via an indie marketplace such as Itch.io

1.3 Major constraints

The game will be developed using Godot and Blender, therefore the constraints on its development are primarily any constraints Godot or Blender may impose, or difficulties that may arise from the developers' relative inexperience with the software. Mouse and keyboard interaction on a Windows PC is the current focus for game interaction context, however gamepad input is planned, time permitting.

1.4 Software Process Model

We are adopting the agile process model. The agile model allows us to prototype the game quickly and make changes as we need to.

2.0 Data Design

2.1 Game State Management

GameControl

Attributes	playerA playerB
Behaviors	_Input() _Ready()

2.2 Player Characters

PlayerA/B

Attributes	acceleration active air_acceleration camera camera_pivot gravity interact_collider inventory jump_power max_pitch max_terminal_velocity min_pitch mouse_sensitivity playerB/A speed velocity y_velocity
Behaviors	HandleMovement() GetDirection() _Input _Process() _PhysicsProcess() _Ready()

2.3 Interactable Entities

InteractableNPC

Attributes	dialogText dialogBox
Behaviors	_on_Item_body_entered()

	_Ready()

DefaultItem

Attributes	
Behaviors	_on_Item_body_entered()

LoadScene

Attributes	sceneToLoad
Behaviors	_on_Item_body_entered()

2.4 In-Game User Interface Entities

DialogBox

Attributes	dialogue[] dialogueIndex finished text
Behaviors	AdvanceDialogue() _Ready()

3.0 User interface design

A description of the user interface design of the software is presented.

3.1 Main Menu (TPT MENU 1-4) [SRS 4.2, 4.3]

- **MENU-1:** Player can start a new game
    - On click of the start new game button in the menu, the player should be placed in the first level of a new game.

- **MENU-2**: Player can load an old saved game
    - On click of the load game button, the player should be taken to another screen that lists the previously saved games.
        - On click of an item on that list, the player should be loaded into the corresponding saved game.
        - If there are no saved games, the load game button should be grayed out and unclickable
- **MENU-3**: Player can exit the game application
    - On click of the exit game button, the application should be terminated and the player should be returned to the desktop.

3.2 Pause Menu (TPT PAUSE 1-2) [SRS 4.2, 4.3]

- **PAUSE-1:** Player can save game
    - On click of the save game button in the menu, the game should display an in-progress indicator until the save game operation is completed.
        - The game state should be saved locally.
- **PAUSE-2**: Player can quit game
    - On click of the quit game button, the player should receive a confirmation prompt with a reminder to save their progress.
        - If the player confirms the quit game action, the player should be returned to the main menu.

3.3 Gameplay (TPT GAME 1-6) [SRS 4.1, 4.4, 4.5]

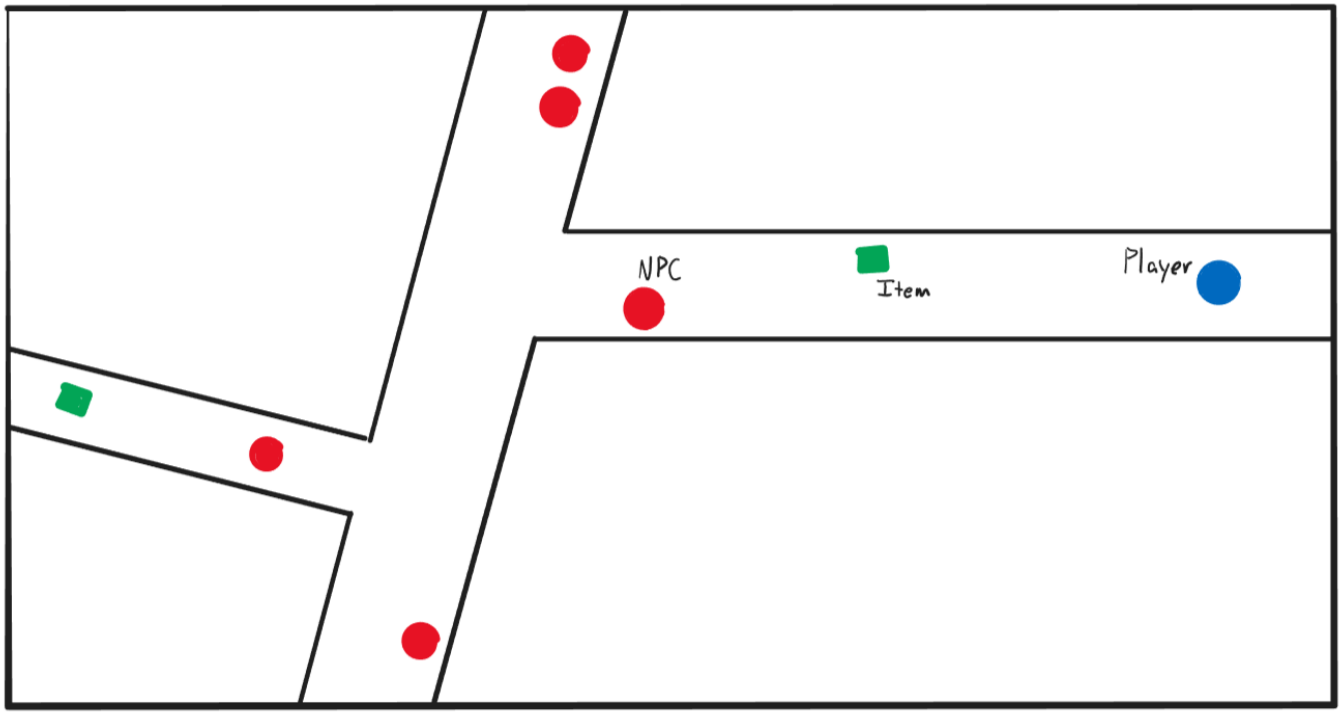
- **GAME-1:** Move with the camera centering on the character [SRS 4.1]
    - ‘WASD’ keys move player in relative space with respect to viewport, ‘Space’ key causes player to jump and subsequently be returned to ground by gravity
- **GAME-2:** Adjust the camera using the mouse [SRS 4.1]
    - Mouse left and right shift yaw and mouse up and down shift pitch, respectively
- **GAME-3:** Collide with game world [SRS 4.1]
    - Player cannot clip through level geometry or fall through floor
- **GAME-4:** Switch between two avatars [SRS 4.1]
    - ‘Q’ key switches control and camera between one of two playable characters, inactive character follows active character
- **GAME-5:** Interact with entities in environment [SRS 4.1];
    - ‘E’ key contextually interacts with interactable entities
    - Some interactions produce inventory items [SRS 4.5];
- **GAME-6:** Travel between separate scenes [SRS 4.4]
    - ‘E’ pressed on LoadScene item transitions to scene specified

3.4.1 Description of User Interface

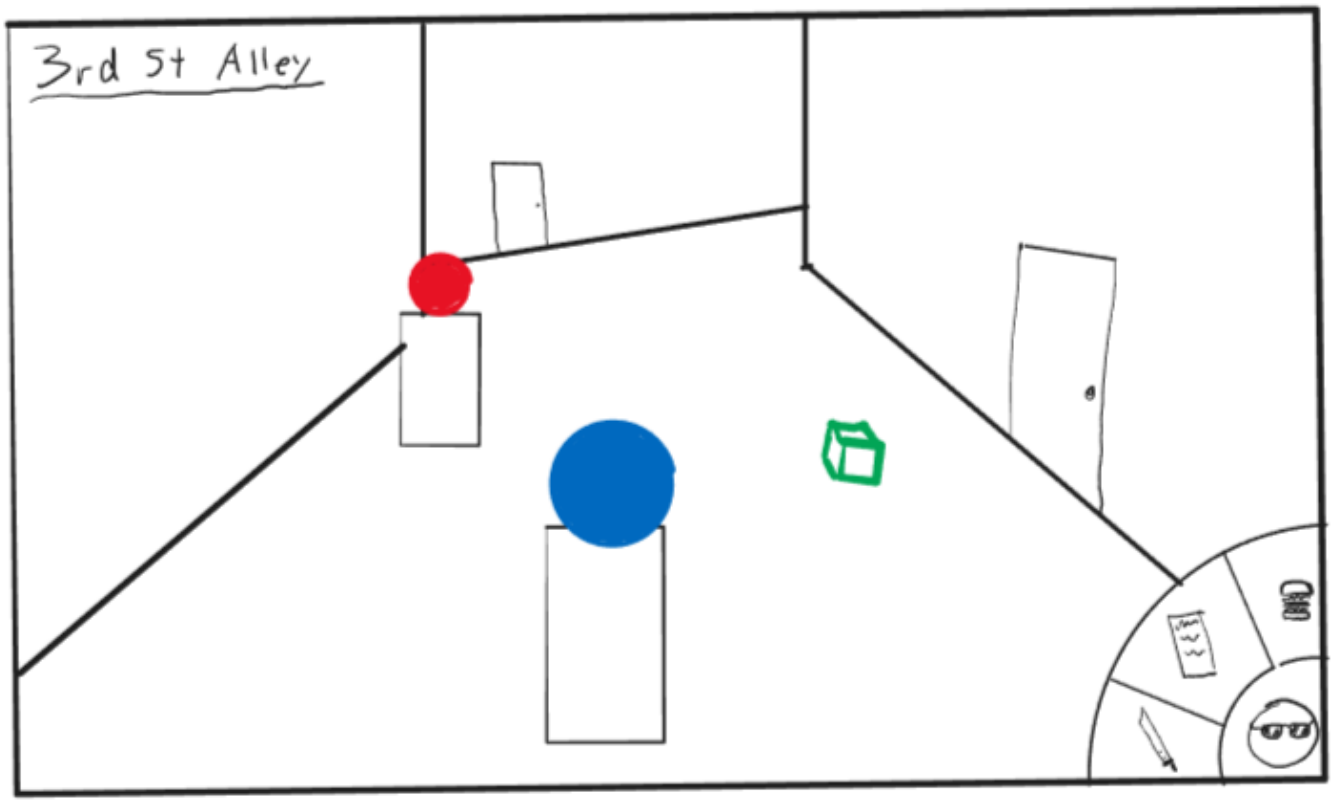
The user will interact with the game via mouse and keyboard, as described above and in the Test Plan (see Appendix A). Relevant information to player such as inventory items and name of current scene will be displayed on screen along with player character and game world.

3.4.2 Screen images

A mockup of in-game scene:



Mockup of player viewport:



3.5 Interface Design Rules

These rules match the design rules for the main menu, but are repeated here for clarity and some sections have been adjusted for relevance to the pause menu in particular.

- **Consistency**
  - Visual components are consistent across menus
  - UI design for navigational/menu items is consistent with the overall visual theme of the game

- **Usability and reduced memory load**
  - The menu components are named or visually coded in an intuitive way
  - Interface is organized hierarchically: the top-most level of the menu contains only the most basic functions, and each sub-level menu details further actions on each of those functions where applicable

- **Place the user in control**
  - There are no buttons the user cannot interact with
  - Critical actions should allow for confirmation to protect user from errors resulting from accidental clicks (such as not allowing the user to quit without confirming that action, and allowing them to go back to save first)
  - All technical in-between steps are hidden from the user
    - Exception: any necessary loading time indicators

4.0 Appendix A: Testing Plan Table

Functional Area	Requirement ID	Method/ Function	Test data/Input	Expected output	Actual output
MENU	MENU-1	Start new game	Left mouse button click or Enter key press over ‘New game’ button	Player is placed in new game or tutorial	
MENU	MENU-2.1	Load saved game	Left mouse button click or Enter key press over ‘Load game’ button	Player is taken to a screen displaying saved games	
MENU	MENU-2.2	Choose saved game to load	Left mouse button click or Enter key press over chosen saved game on the list	Player is loaded into the selected saved game	
MENU	MENU-2.3	Return to main menu	Left mouse button click or Enter key press over ‘Back’ button in saved games list screen	Player is returned to main menu	
MENU	MENU-3.1	Exit game application	Left mouse button click or Enter key press over ‘Exit game’ button	A confirmation message is displayed	
MENU	MENU-3.2	Confirm exit game application	Left mouse button click or Enter key press over ‘Exit’ button	The game application terminates	
MENU	MENU-3.3	Cancel exit game application	Left mouse button click or Enter key press	Player is returned to the main	
MENU	MENU-4	Return to main menu	Left mouse button click or Enter key press over ‘Back’ button in high scores list screen	Player is returned to main menu	
PAUSE	PAUSE-1	Save game	Left mouse button click or Enter key press over ‘Save game’ button	Game data is saved and a message is displayed confirming it	
PAUSE	PAUSE-2.1	Quit game	Left mouse button click or Enter key press over ‘Quit game’ button	A confirmation message is displayed	

PAUSE	PAUSE-2.2	Confirm quit game	Left mouse button click or Enter key press over ‘Quit’ button	Player is returned to the main menu	
PAUSE	PAUSE-2.3	Cancel quit game	Left mouse button click or Enter key press over ‘Cancel’ button	Player is returned to the pause menu	
GAME	GAME-1.1	Input: move forward	Keyboard ‘W’ pressed	Player moves forward, away from viewport	
GAME	GAME-1.2	Input: move backward	Keyboard ‘S’ pressed	Player moves backward, toward viewport	
GAME	GAME-1.3	Input: move left	Keyboard ‘A’ pressed	Player moves to the left, parallel to viewport	
GAME	GAME-1.4	Input: move right	Keyboard ‘D’ pressed	Player moves to the right, parallel to viewport	
GAME	GAME-1.5	Input: jump	Keyboard ‘Spacebar’ pressed	Player jumps into air and is grounded by gravity	
GAME	GAME-2.1	Input: look up	Mouse moved up	Viewport pitch shifts toward sky	
GAME	GAME-2.2	Input: look down	Mouse moved down	Viewport pitch shifts toward ground	
GAME	GAME-2.3	Input: look left	Mouse moved left	Viewport yaw shifts to left	
GAME	GAME-2.4	Input: look right	Mouse moved right	Viewport yaw shifts to right	
GAME	GAME-3.1	Horizontal world collision	Move into solid object	Player stopped by solid object	

GAME	GAME-3.2	Vertical world collision	Fall into floor	Player stopped by floor	
GAME	GAME-4	Switch player	Keyboard ‘Q’ pressed	Player control and camera switch from active character to inactive player character (A to B or B to A) and newly inactive character follows active character	
GAME	GAME-5.1	Interact with objects	Keyboard ‘E’ pressed	Object _on_body_entered() triggered	
GAME	GAME-5.2	Advance dialogue box	Keyboard ‘E’ pressed while dialogue is on-screen	If there is still dialogue to display, show next line, otherwise, close and reset dialogue box	
GAME	GAME-6	Load new scene instance	Keyboard ‘E’ pressed on LoadScene item	New environmental scene is instanced around player	

5.0 Appendix B: RMMM

RISK	PROBABILITY	EFFECTS
1) Difficulty adjusting to new software and toolchain (Godot, Blender)	Low	Tolerable
2) Team member illness	Moderate	Tolerable
3) Lack of time-management	Moderate-high	Serious
4) Disagreements between team members about development	Low	Tolerable
5) Change in game concept / scope	Moderate	Serious
6) Lose progress/files	Low	Serious

RISK	STRATEGY
1) Difficulty adjusting to new software and toolchain (Godot, Blender)	**Mitigation:** Choose tools with excellent documentation and community support that are extensible and allow for non-destructive changes in workflow or toolchain. **Monitoring:** We will regularly check in with each other on our progress and comfort level with our chosen toolchain **Management:** Assets and basic systems for our tools can be imported to aid with development in the event of disruption to schedule or time crunch. Certain tools can be switched out with tools we are more familiar with largely non-destructively
2) Team member illness	**Mitigation:** Shared responsibilities among team members will mitigate effects of temporarily reduced team size, no one will be singularly responsible for any element. Time requirements will be over-estimated to ensure slack time can support task reorganization. We will be reasonably careful with exposure to illness **Monitoring:** Regular team check-ins and otherwise good communication will allow us to plan ahead if we anticipate someone will fall ill **Management:** Team members will be re-assigned to tasks that can no longer be completed by ill team member(s), there are at least two people on each task to ensure smooth transitions in this eventuality. If both team members on a particular task fall ill, our tools are well-documented and shouldn't be particularly difficult to get up to speed with for the remaining team members
3) Lack of time-management	**Mitigation:** Take advantage of slower periods in

	homework. It seems when one class assigns homework, the rest will fall in similar intervals. It'd be best to consistently take advantage of those slower periods to plug away on the project **Monitoring:** Compare project progress to class project timeline, if project milestones are not completed in pace with expectation then this risk may be realized **Management:** Similarly to handling team member illness, multiple people are stationed on each task such that slack time can be repurposed to increase development pace in problem areas
4) Disagreements between team members about development	**Mitigation:** Check in with everyone throughout the project to make sure everyone is still in agreement or has any concerns. **Monitoring:** Good team communication to anticipate and handle disputes before they become problematic **Management:**Try to find some compromise.
5) Change in game concept / scope	**Mitigation:** Concept has been scaled down to MVP to simplify systems such that scope should not need to change. **Monitoring:** If existing game design proves too large in scope, unimplementable, or otherwise causes issues or is simply not fun, then a change in concept or scope may be necessary **Management:** In the event that scope does change necessarily or by choice, our base systems should still be usable, with few components needing to be updated. If scope is increased, additional content can be built off of existing scenes and assets.
6) Lose progress/files	**Mitigation:** Share files with each other, most likely through Github. Handle merge conflicts. **Monitoring:** Make sure everyone has access to all files at all times. Constantly save/share any new progress. **Management:**.Help each other recover what we can.

6.0 Final Notes

Member Roles

- It should be noted that the roles set for the team members are not strict. We will certainly all make story and design decisions and we will all most likely be writing some code