# Custom Program Design Report

Name: Andrew Teck Foon YII

Student ID: 104386568

1. Provide a summary of your program, maybe write up a paragraph or two to describe what does it do? What are some of the key features etc?

I have developed a 2D platform farming game for my custom program in C, involving the use of raylib for game development and json-c for data processing. It contains six screens that will be changed according to the respective gameplay mode: **Menu Mode**, where the player can start or exit the game and control settings related to sound. **Loading Mode**, where the player decides to start a new game or load one that was saved (Both proceed to the **Playing Mode**). **Setting Mode** allows the players to go into'*How To Play*', '*Save The Game*', '*Back To Main*', or control settings related to sound. (After pressing the setting button in **Playing Mode**) **Instruction Mode**, where the player can view character controls by selecting the '*How To Play*' in the **Setting Mode**. After pressing *'Save Game*' in the **Setting Mode**, a small pop-up confirms that the game has been saved. While pressing *'Back to Main*' in the **Setting Mode**, the player will navigate back to the **Menu Mode**.

Moreover, press the cross button on the right-hand side in the **Setting Mode** to return to **Playing Mode**. **Playing Mode**, where the player controls the character, farms crops, waters the plants by preventing their death, and harvests and sells them to gain money. Selling and price negotiation have also been applied; the player can control their character to share and negotiate the sale of crops grown with the seller. **Ending Mode**: when the player's earnings of more than 1000 coins have been achieved, they will have completed the game. The game will then proceed with the credits page, and an option to continue playing will pop out, prompting the user if they would like to continue playing and enjoying the game further even though they have completed it.

1. Drawing a picture of what you want it to look like, sketch the first few interfaces that will be shown when the program is run.

**Draft Drawing of Each Screen**

button

2D Farmer

Simulator

music

**Quit**

**Start**

Menu Mode Screen

**New Game**

close

**Load Game**

button

Animation of the game

music

Loading Mode Screen

Character

Item Slots

button

Money:

Harvested Wheat:

setting

Playing Mode Screen (Fixed)

Character

Seller

Paddy Field

Home

Playing Mode Screen (Map)

Speaker Icon

Chatbox

Character

Message…

Seller

Playing Mode Screen (Chat Box)

button

Save Game

How To Play

Back To Main

close

music

Setting Mode Screen

button

Info……

close

music

Instruction Mode Screen

button

music

Picture

Saving Message…

Saving

Keep Going-Up Game Credit…

Short video about gameplaying

Ending Mode Screen

**Real Outcome of Each Screen**

A screenshot of a video game

Description automatically generated

button

Menu Mode Screen (Advanced with Background Animation)

A video game screen capture

Description automatically generated

button

Loading Mode Screen（Change the button format so that it provides different visual effect）

A screenshot of a video game

Description automatically generated

Playing Mode Screen (Fixed Icon + Map)

A video game screen with a cartoon character

Description automatically generated

A video game screen with a cartoon character

Description automatically generated

Playing Mode Screen (Chat Box)

A screenshot of a video game

Description automatically generated

Playing Mode Screen (Paddy Field, Seed, Water, Harvested, Dig, Different Growth Rate Picture)

A screen shot of a game

Description automatically generated

Setting Mode Screen

A screen shot of a game

Description automatically generated

Instruction Mode Screen



Saving

A video game screen with a cartoon character

Description automatically generated

A screenshot of a video game

Description automatically generated

Ending Mode Screen

1. List and describe some of the main data types:

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Description | Example of value |
| screenWidth | int | A global constant variable that sets the screen width of the game. | 1500 |
| screenHeight | int | A global constant variable that sets the screen height of the game. | 1000 |
| savedfilename | char\* | A global constant string variable to store the path of the save file. | "saving/saveddata.json" |
| rawfilename | char\* | A global constant string variable to store the path of the new game file. | "saving/rawdata.json" |
| credits | char\*[] | A global constant string array to store information, which will be displayed in the game credit frame (**Ending Mode**) | {  "Game Developed By:",  "Andrew Yii",  "",  ……,  NULL  } |
| GameMode | struct (enum) | A custom enum struct to control the display of six different screens | {  MENU,  LOADING,  SETTING,  INSTRUCTION,  PLAYING,  ENDING  } |
| Chat | struct | A custom struct to manage the display of chat content in the game. The fields in the struct are as follows:   * **id (int)**: Indicates the sequence or order of the chat content. * **content (char [256])**: Stores the chat message or dialogue text. * **whosay (int)**: Specifies who is speaking the message (e.g., 0 for Player, 1 for NPC). * **showoption (bool)**: Determines whether options (choices) will be shown to the player during the chat. * **accept (bool)**: Determines whether the player accepts or rejects an offer made by a seller. * **caninput (bool)**: Specifies whether the chat content allows the player to input a response. | {  int id;  char content[256];  int whosay;// 0(for seller), 1(for player)  bool showoption;  bool accept;  bool caninput;  } |
| slot | struct (enum) | A custom enum struct to control the selection of items in the item slots designed. The enum defines the different actions of players. | {  slot0,  slot1,  slot2,  slot3,  slot4,  slot5  } |
| GameState | struct | A central container for all the key data in the game. Organising all game-related information into a single struct makes it much easier to pass and manage the game state across different parts of the program instead of passing multiple individual variables (like player data, camera settings, and game mode) separately. | {  GameMode mode;  Texture2D player;  Camera2D view;  int array\_land[5][6];  Rectangle playerSrc;  Vector2 origin\_player;  Bool playerMoving;  Color bgcolor;  Font myfont;  Music bg\_music;  Sound dig\_sound;  float playerspeed;  ……  } |
| currentFrame | int | A local variable under function is used to change the frame to display the animation effect. | 0 |
| MenuFrameTime | float | A local variable is used to detect the time change and make the change of frames. | 0.1f |
| elapsedTime | float | A local variable that keeps changing and is detected by MenuFrameTime is a variable for frame changing. | 0.0f |

1. Describe the main functions and procedures. Have enough that you can start to see how the program will continue to develop as you progress. (e.g.: initialize, draw, update, etc):

|  |  |
| --- | --- |
| Function /Procedure/Method | Description |
| void SetupCustomTheme() | Customises input UI appearance by editing styles from the Raygui library, which provides a look that would fit the game's aesthetic. |
| void UpdateDialog(Gamestate \*state, int situation) | Updates the contents in the chatbox dynamically according to the situation number, which decides what it needs to present during the gameplay conservation. |
| void Initialise(Gamestate \*state) | Loads all essential resources such as textures, fonts, music, and sounds. It also initialises the camera (Camera2D) and sets default game values, e.g., state->Mode is set to MENU to indicate the starting mode. |
| void moveAllY(Gamestate \*state, int delta) | Change all in-game icons Y to maintain the player's movement. This keeps on-screen UI elements constant while the player moves up and down the screen. |
| void moveAllX(Gamestate \*state, int delta) | Moves the X-coordinates of all game icons to match the player's movement on the horizontal axis so that the UI remains in synch with the player's position. |
| void CollisionObject(Gamestate \*state) | Checks for collisions between the player and any other structure present in the game scene so that none pass through each other in an unrealistic manner. |
| float clamp(float value, float min, float max) | Restricts a given value within the specified minimum and maximum boundaries. This function is used to limit the player's movement within predefined areas. |
| void Chatting(Gamestate \*state) | Checks the distance between the player and the NPC (e.g., a seller) to activate chat interactions when they are within a specific range. |
| void ConstructScene(Gamestate \*state) | Builds and renders the content of the main gameplay screen (PLAYING mode), including all interactive elements and the environment. |
| void DrawMenu(Gamestate \*state) | Creates and displays the MENU mode, outlining the main navigation options, such as start, exit, and music control. |
| float save\_game(float value,……) | Saves the game state to a JSON file using json\_object\_add to serialise data and fprintf to write the file. This allows players to retain progress for future sessions. |
| void DrawSettingMenu(Gamestate \*state) | Structures and displays the settings menu where players can adjust options such as how to play, save the game, and return to the main. |
| void load\_game\_date(const char\*referfile, Gamestate\*state) | Loads saved game data from a specified JSON file and adjusts game conditions to reflect the previously saved state, ensuring seamless continuation. |
| void DrawingLoadingMenu(Gamestate \*state) | Renders the loading screen with the new game/load game options. |
| void DrawingInstructionMenu(Gamestate \*state) | Structures and shows the instructions or help screen, guiding players on gameplay mechanics and controls. |
| void DrawingEndingMenu(Gamestate \*state) | Displays the ENDING mode, showing the final screen with a summary and credits after gameplay completion. |
| void SetupWindow(Gamestate\*state, int windowcase) | Configures the game window about different screen types in different scenarios to ensure proper layout and drawing. |
| void Instruction(Gamestate\*state) | Keyboard input detection. Controls the player through actionable feedback, such as moving the character, interacting with objects, or opening the main menu. |
| void actionactivate(Gamestate\*state) | Manages the item slot selection and triggers the action corresponding to the activated item. Improves interactivity on the game front. |
| void GardeningAction  (Gamestate\*state, int number) | Determines the direction and conditions for performing gardening (digging) actions in the game, responding to player input. |
| void Gardening(Gamestate \*state, float Currentplayerx, float currentplayery) | Checks the player's position to activate gardening activities within a suitable range, allowing for realistic interaction. |
| void Seedling(Gamestate \*state, float Currentplayerx, float currentplayery) | Detects the proximity in order to turn on seedling actions; enables players to plant seeds when conditions are met. |
| void WateringAction  (Gamestate\*state,int number) | Determines the direction and application of the watering action based on the player’s direction. |
| void Watering(Gamestate \*state, float Currentplayerx, float currentplayery) | Check the proximity of the player to trigger the watering action when standing close enough to a plant. |
| void HarvestingAction  (Gamestate\*state, int number) | Handles initiation of harvesting actions based on the player’s direction. |
| void Harvesting(Gamestate \*state, float Currentplayerx, float currentplayery) | Trigger processes involved in harvesting when the player is in range and allows them to collect grown crops. |
| void Growth(Gamestate \*state) | Implements the mechanism for plant growth over time, which includes water depletion. This simulates realistic growth cycles and resource management. |
| void LeaveGame(Gamestate \*state) | Unloads all game resources such as textures, fonts, music, and sounds to free up memory and prepare for game exit. |
| void cursorchanging(Gamestate \*state, int casetype) | Manages cursor visibility and behaviour, displaying a custom game cursor and activating hover animations when applicable. |
| void main() | Acts as the entry point for the game, initialising core functions, entering the game loop, and handling overall game flow. |

1. Flowchart of my program.

**Flowchart 1**

A diagram of a flowchart

Description automatically generated

[Overall Gameplay Flowchart](https://studentsswinburneedu-my.sharepoint.com/:u:/g/personal/104386568_students_swinburne_edu_my/EYu9doC-j-9Lmrd0ohbMu5gBGxOasMKOVhJX5Pfi2JA1ZA?e=TSzVda)

(You may press it to see more details because the picture is unclear. Using the Swinburne account).

**Flowchart 2**

A diagram of a flowchart

Description automatically generated

[PLAYING MODE](https://studentsswinburneedu-my.sharepoint.com/:u:/g/personal/104386568_students_swinburne_edu_my/EcLsbcv3PHxOqpxVGVLNCfIBRhxkF2Je0MIlJ1Rv1HqSqA?e=17egCl)

(You may press it to see more details because the picture is unclear. Using the Swinburne account).