TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

SOFTWARE ENGINEERING METHODOLOGIES

MAINTENANCE DOCUMENT GROUP 3

.

1. BASIC INFORMATION

The software is a 2D game made with Unity 2017.3.0f3. The game consists of a number of levels in addition to the menu screens. The game is modelled after the 1982 game Jungle Hunt. This chapter gives an overview of the software structure.

The game starts with menu screens. During these screens the difficulty and the players name are saved into the DataContainer_Character script. This script holds all the information related to the player's success in the game: current level, scores and the number of lives.

Each level is generated dynamically using the corresponding LevelGenerator script. This script uses prefabs and graphics with the information saved into DataContainer_Character to create the level. Renaming or changing graphics or prefabs should be changed accordingly in the generator scripts.

The Player-script is used to move the player and change the players state and animation. PlayerCollision takes care of the different collision and triggers that happen when the player hits non-lethal objects.

The game uses Unity's Input Manager. This ensures that the game is playable with different control schemes and developers can change the bound buttons easily. The menu's can be navigated with mouse, keyboard or another device. Player can be controlled using keyboard or similar device.

The software has built-in unit tests for SceneManager and DataContainer_Character scripts. These tests can be run be using the Unity's own Test Runner. In addition to these tests, the software has been tested manually. In normal use the game should run as intended.

Due to time constraints there are some known bugs in the game:

- Hitting two ropes at the same time softlocks the game, ie. the player cannot do anything.
- Pressing quit multiple times in the main menu causes the sound to be played multiple times.
- If the player hits the bubble the same time as the bubble hits the surface, the player will become helpless and the game continues only after the air runs out.
- PlayerLivesTests unit test may not always work if you run playmode tests before.

2. MORE DELICATE PARTS OF THE SOFTWARE

2.1 Scene_Manager

Scene_Manager (note that this is not the same as Unity's built-in SceneManager) is a large part of preserving continuity in the game. It handles Scene Transitions and the general flow of things within the game.

Scene_Manager is hosted in an "Overlay Canvas" object which allows it to draw UI objects on screen without need for camera references or other constraints. This allows each Scene to be designed uniquely while still maintaining similar features, such as UI.

Scene_Manager is responsible for maintaining the following main game features:

- Sound (via SoundSystem component)
- User Interface and Transitions.
- Input Events (using Unity's EventSystem)
- Loading levels and providing ways to load levels.
- Generating and Maintaining Level Order.

The Scene_Manager is generated on boot-up when loading to the main menu. An additional check is performed to ensure no additional copies are created. The manager only has two main functions to call: ChangeScene(int scene) and NextLevel(bool scoreboard) the former allows any scene to traverse to another without regard to player condition etc. the latter allows levels to call the next level without caring what level is coming next. The following is an outline of main functions that are travelled through when changing scenes:

Next Level – *Generates a level order if not existing and calls the next one.*
Change Scene – *Starts change to next scene, starts by initiating a transition event to which the following is always subscribed to.*
[Scene Change Component - Image] **Drop Image** – *Drops a transition screen.*
[Scene Change Component - Image] **Score Board Completion** -> **Button Call** – *If scoreboard Boolean is set to true, a scoreboard is shown and a button to continue given, otherwise skipped.*
[Scene Change Component - Image] **Wait For Completion** – *Begins an asynchronous load and waits for it to finish, then raises the transition screen.*
Start Load – *Called when Load starts, used for fade outs, hiding UI etc.*
Finished Load – *Called when Load finishes, used to play new BGMs and show UI elements etc.*

Start Load and Finished Load functions have various conditional checks that allow it to hide, show UI components and play the correct BGM for each level. The entire system uses Unity's BUILD INDEXES to determine what screen is being presented, including the Change Scene function.

2.2 Player

The Player script is responsible of the player characters controls and state changes. Closely related to the Player script is the PlayerCollision script that handles the different collision and triggers that the player may encounter.

The Player script manages:

- Player states
- Player controls
- Changing players animations

The Player-script holds different values that are used to move the player. Most of these values can be changed directly from the Unity editor. The XAxis and YAxis values change the forces affecting the players jump and the Speed value is used to change the players speed.

Controlling the player happens through as State machine. Certain actions are only permitted in certain states. This ensures that the player cannot jump repeatedly or start running while they are underwater. Each state has a priority that corresponds their position in the state definition, ex. State_Dead has the highest priority. This is to ensure that the player doesn't get caught in a state they cannot recover from. When player state changes, the corresponding animation is also triggered.

Player script has also functions DeadlyHazard, InTheBubble and OutOfBubble. These functions are called when the by scripts in the Bubble and enemy Prefabs. When the players state changes to dead, it triggers a function that resets the level if there are still lives left, otherwise it ends the game.

The update function keeps tab on the pressed buttons, and if the button corresponds to a valid input (within the player state) it executes the corresponding function. In practice, the ChangeState is always waiting for the correct input.

The PlayerCollisions is a separate script, but in theory it could be incorporated to the Player script. It controls

- Level end triggers
- Rope catching
- Returning to proper state and animation after jumping

The triggers that are used to change levels and the colliders used to change state use the same names as the objects in LevelGenerator scripts.