VOYAGE

Technical Design Document

May 2019

Contents

Game Overview

Game Developers:

Irina Brigido Chan

Armando Perez

Anthony Reese

Dwayne Wilkes

Game Title: Voyage

Game Platform: Mac OS10.11+ and Windows 7 SP1+

Game Style: Third-Person Action Exploration

Rijeka Repository: CS596S19_3D_Game_Team04

URL: https://tinyurl.com/y482yhe3

Game Overview

Voyage is a third-person Action Exploration game where the player is a space traveller who crashed into planet P-596. The player needs to find the three lost ship pieces and deliver them back to the main ship in order to successfully complete the game. There are different types of enemies in the environment that the player will need to avoid, as well as items that replenish a small portion of the player's health.

Technical Summary

The game was developed using the Unity game engine (version 2018.3.x). Blender for 3D asset creation, and Adobe Photoshop CS5.1 for texture painting and UI. The game was developed for PC and MacOS simultaneously, and uses the basic Unity System Requirements[1] for development and final build.

[1] https://unity3d.com/unity/system-requirements - Final requirements may vary.

Version Control

The game was developed by keeping a working version of the project on the `(master)` branch, with coordinated effort among the developers for minimizing merge conflicts. For modular implementation of features and to assist with concurrent development, each team member used a separate branch (when applicable) and merged back to the master branch when the artifact was complete.

To help with consistency, the following guidelines for Git commit messages and branching were used:

- https://tinyurl.com/lgjkjsk
- https://nvie.com/posts/a-successful-git-branching-model/

Development Plan

Product	**Prototype**	**Final Build**
3D Game Assets	Simple geometric shapes as placeholders for items, ship prefabs and enemies	"low-poly" style assets will replace placeholder objects
Environment	Main terrain mesh with few environment items for easier development	Populated environment with diverse terrain modifiers and vegetation
Scripting	Basic character controller with simple camera follow Basic enemy AI behavior Few ship pickup items, few spawn points, static ship model.	Optimized camera follow Optimized AI behavior with additional modes More spawn points, responsive ship model to player item retrieval
Audio	No audio	Audio effects

Milestones

4/30 - Prototype delivery

5/7 - Final build

Project Tracking and team communication

Slack was used for team communication. Scrum meetings were registered on the Wiki section of a separate Rijeka repository: https://tinyurl.com/y5n4p46e

The Issue and Board sections of the repository were used for tracking bugs and features that needed development or improvement.

Script Summary and Analysis

Below is a description of the scripts included in the project, along with Big O analysis. All scripts perform in O(1) (constant time) as there are only conditional statements being used mainly by Update(), Awake() and Start().

Player:

- *Player.cs:* Tracks maximum and current health of the `PlayerCharacter GameObject`. Checks on `Update()` if `currentHealth` is equal or below to 0 in order to load `GameOverScene`.
- *PlayerInputController.cs:* Listens to user's key input to be used by *PlayerMotor.cs* for controlling the character.
- *PlayerManager.cs:* Ensures there is only one instance of a `GameObject PlayerCharacter` in the scene.
- *PlayerMotor.cs:* Script that moves the `PlayerCharacter` and translates the camera accordingly. `Update()` checks if there is collision between the player and the ground to handle jump correctly.
- *PlayerAnim.cs:* Controls running and idle animations.

Camera control:

The following scripts control the camera behavior. The scripts perform in constant time as it only listens to user key input inside the `Update()` function:

- *FollowTarget.cs*
- *ThirdPersonCamera.cs*

Enemy and AI behavior:

- *EvasiveController.cs:* Controls the behavior of the ranged, dynamic enemy. `Update()` detects if the player is within the range set by `lookRadius`. If too close, the enemy runs away. Attack speed and ratio set by the variables `projectileSpeed` and `coolDown`. Enemy speed set by `enemySpeed`. Finally, `shootRadius` sets the range within the enemy shoots projectiles at the player.
- *ProjectileController.cs*: Instantiate a projectile (from Prefab) and shoots in the direction of the target when target is within range. Range set by `lookRadius`, shooting speed and ratio set by `projectileSpeed` and `cooldDown`, respectively.
- *DestroyProjectile.cs*: Destroy the projectile instance when detecting collision in `OnCollisionEnter()` with GameObjects tagged with `Player` or `Enemy`.
- *SkeletonController.cs*: Controls the behavior of the Skeleton enemy (melee, dynamic). Sets damage value on `atkDamage`. Animation characteristics are set by the variables `runningSpeed` and `attackingSpeed`. `Update()` detects if player is within the range set by `lookRadius,` and controls running and idle animations. If reaching the player's current position, `Update()` sets the attacking animation to true. `OnCollisionDetection()` listens to incoming damage from projectiles.

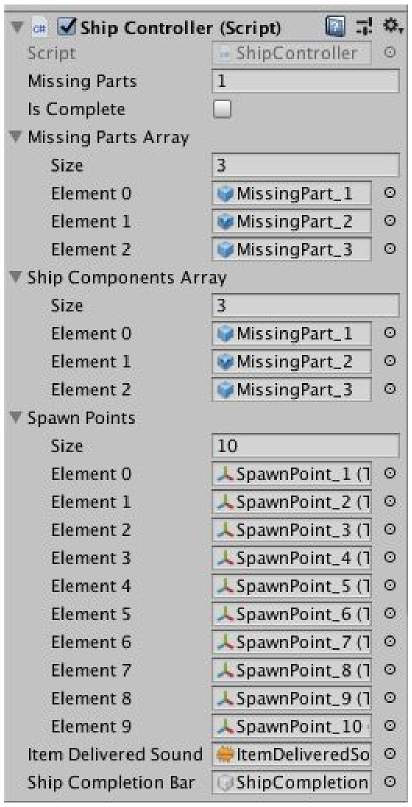
Game and Items

- *GameManager.cs*: Ensure only one instance of the `GameManager GameObject` exists at the scene. Listens to scene loading through `OnEnable()` and `OnDisable()` methods in order to display the appropriate cursor (crosshair when in game, regular cursor when in menu). `Update()` listens to user input as the key "Escape" quits the application.
- *powerUpItemsController.cs*: Controls the health pickup items placed in the scene. `Update()` animates the items (rotation over time). `OnCollisionEnter()`

detects collision with the `PlayerCharacter`. Then, a value is randomly generated within a range (set between 5 and 15).

> *Note:* The player cannot have `currentHealth` above the maximum health set by *Player.cs.* Picking up a health pack while having full health will maintain the player's health at maximum, therefore having no impact. Similarly, picking up a health pack that has a value generated which would boost `currentHealth` to be above the maximum health will cap the player's health at the maximum value set by *Player.cs.*

Ship



```
▼ ✓ Ship Controller (Script)
   Script                  ShipController
   Missing Parts           1
   Is Complete             ☐
▼ Missing Parts Array
     Size                  3
     Element 0             MissingPart_1
     Element 1             MissingPart_2
     Element 2             MissingPart_3
▼ Ship Components Array
     Size                  3
     Element 0             MissingPart_1
     Element 1             MissingPart_2
     Element 2             MissingPart_3
▼ Spawn Points
     Size                  10
     Element 0             SpawnPoint_1 (T
     Element 1             SpawnPoint_2 (T
     Element 2             SpawnPoint_3 (T
     Element 3             SpawnPoint_4 (T
     Element 4             SpawnPoint_5 (T
     Element 5             SpawnPoint_6 (T
     Element 6             SpawnPoint_7 (T
     Element 7             SpawnPoint_8 (T
     Element 8             SpawnPoint_9 (T
     Element 9             SpawnPoint_10
   Item Delivered Sound    ItemDeliveredSo
   Ship Completion Bar     ShipCompletion
```

- *ShipController.cs*: Controls the completion status of the spaceship. Instantiates ship pieces at the spawn points provided, ensuring only one part is placed at a location. Once all parts are delivered, `SceneManager` loads the `CreditsScene`. `OnCollisionEnter()` detects collisions with the `PlayerCharacter` and further checks if there are collected parts ready for delivery.
- ShipPartsController.cs: Script attached to each ship part prefab. Has a boolean variable `collected` that is set to true once a collision with the `PlayerCharacter` is detected with `OnCollisionEnter()`.

Menu and UI

- *CrossHair.cs*: Draws the provided crosshair at the center of the screen.
- *HealthBar.cs*: Controls the graphical bar on the UI that reflects the current health of the Player. Obtain values from *Player.cs* and scales the object so that `1f` displays a full bar, `0f` displays an empty bar, and values in between are scale the health bar appropriately.
- *MenuController.cs*: Controls loading of different scenes and closes application.