Team Playfetch

GYPSY'S TALE - TECHNICAL DESIGN DOCUMENT

21 JULY 2018

Document version 1.0
Author: Thomas Wiltshire

TABLE OF CONTENTS

TABLE OF CONTENTS 2

PROJECT OVERVIEW 3

- GAME CONCEPT 3
- INTRODUCTION 3
- GAME ENGINE 3
- FEATURES 4
- TECHNICAL RISKS 5
- ART REQUIREMENTS 6
- THIRD PARTY TOOLS 7
- SYSTEM REQUIREMENTS 8

GAME SYSTEMS 9

- OBJECTS, SCRIPTS AND SYSTEMS 9
- CLASSES 15

USER INTERFACE 16

- MENUS AND GAME UI 16
- INPUT METHOD 17

CODE OVERVIEW 19

- CODING CONVENTIONS 19
- COMMENTING 20
- UNITY ATTRIBUTES 21
- CODING GUIDELINES 22
- SOURCE CONTROL 22

PROJECT TEAM 23

- MEMBERS 23
- TEAM SIGN-OFF 23

PROJECT OVERVIEW

GAME CONCEPT

Third person adventure narrative game based around the journey of a lost dog trying to find their way home. The story will be told over (3) days in a giant park environment, where the player will be able to complete little tasks on a “to do” list until their owner finds them.

INTRODUCTION

The gameplay of Gypsy’s Tale will consist of controlling a quadruped dog player with a 3rd person camera around a park to complete various simple objectives. The dog can move around the park and jump, as well as interact with different objects around the map, bark, sprint, and bring up the objectives list.

GAME ENGINE

The game engine of choice for this game is Unity, sometimes known as Unity3D. Unity is a cross-platform engine developed by unity technologies, it can be used to create simulations, two-dimensional and three-dimensional games, virtual reality experiences and so much more.

As this project is a student project under AIE we have decided Unity as our game engine, as there will be plenty of support from teachers, students and online forums, etc. Everyone in the team is already fairly familiar with unity making that another great reason to use it, but most importantly it works perfectly for our game idea because of its handling and support of physics and collisions. Gypsy’s tale is a very visual experience and unity is great for this kind of project.

FEATURES

- **Quadruped dog player controller:** The player can move forward, left, right, backwards and jump, turning of the player will be simply tilting the character. The animation state machine of the player will contain walk, sprint, jump, shuffle (A little shuffle on the spot, used for turning the player when not moving), and howling animation.
- **3rd person camera:** The camera we will be using for our game will be the Cinemachine camera used by unity for the Adam animated shorts. This camera allows us to do cutscenes easily, have different zoom level, nicely flow above or with the player all out of the box with easy setup and use. Using a third party camera means that we can focus efforts on other things like player controllers, animations and the game world.
- **Game world:** The world in which the player will inhabit within the game is a simple park. The park will be inspired by inner-city parks like central park in new york for example.
- **Object interaction:** Throughout the park, there will be multiple interactive objects or objective for the player to interact with. Some of these being the simple pressing of a button and others triggered by location or cutscene.
- **Objectives:** The game will feature many different simple objectives scattered throughout the park, expanded on further below.
- **Howling:** Simple playing of audio from a button press, it will also act as a type of interact button.
- **Day System:** The game will run over 3 in-game days, these days will go for about 8-10 mins each and will determine which objectives can be complete. Each day will have different objectives, if they aren't complete before the end of the day they can't be completed again until the game restarts. The player is required to return to their bed at the end of each day to start the next day.

Objectives:

- **Hot Dog:** The hog dog objective will require the player to find a hot dog wrapper on the ground in the level and interact with it. This will lead the player to a hot dog stand where they can steal a hot dog by interacting again.

- **Learn to Howl:** The player will be able to find another dog in the level that can be interacted with resulting in teaching the player about the howl mechanic. This will trigger a type of music rhythm mini-game requiring the player to press a button at the correct time.
- **Frisbee:** 2 NPCs will be playing frisbee together somewhere in the park. The player will have the ability to interrupt this game and steal the frisbee, the NPCs will then chase the player for a few seconds.
- **Hot Dog Vendor:** There is a hotdog vendor somewhere in the park which the player can steal hot dogs from again. This time, however, the player will be required to knock over 3 bins to distract the NPC that runs the hotdog stand.
- **Busker:** Similar to the ‘learning to howl’ objective the player will be required to complete another rhythm mini-game, however, a little more difficult than last time.
- **Butterflies:** There will be a small butterfly found somewhere in the park, if interacted with the player will eat it. Eating the butterfly activates a quest where the player has to eat each butterfly that appears before the timer ends.
- **Hotdog Sprinkler:** Another hog dog stealing objective, this time the distraction is turning on a sprinkler system.
- **Fetch Quest:** There will be an NPC mother walking around on a set path looking for her child. If you find the child it will follow you back to the mother to complete the objective.
- **Photobomb:** This objective is to attempt a photobomb by jumping into the frame at the right moment of a photo being taken. There will be different lights and sounds from the camera to try and get the timing right.

TECHNICAL RISKS

- **Player Controller:** Being a quadruped there could be possible issues with getting the player nice and polished and working exactly how we are imagining it. This isn’t exactly a risk when thinking of this alone, but when taking into account all of the other things that need to be complete in this game then this a truly real risk.

- **Getting the Camera to feel right:** Even though cinemahine is being used for this project and a lot of the main camera mechanics are already complete there is still the issue of getting the camera to feel right with the player. Not the biggest issue but an issue nonetheless.
- **Day System:** Days in this game are going to determine which objectives can be completed, if not completed can affect the ability to do other objectives on other days. This could become quite complicated depending on how many objectives, how objectives affect others, and what objectives there is to complete.
- **Possible risks with object interactive objectives:** The amount of objectives and interactable objects as well as the complexity of these objects and objectives is a very real risk for this project on whether everything will be completed on time or not.

ART REQUIREMENTS

The graphics formats that will be used throughout the development of the project are listed below with descriptions, file type and why we are using it for our project:

- **FBX:** If you need scene information (Such as light sources for example) or animation, fbx is always the best format choice. It includes the animation, mesh, skeleton, morphs and vertex animation. fbx, however, has a much higher file size compared to the obj format, this is because fbx retains higher-fidelity data and works more efficiently in certain situations. This format is also handy when importing meshes into the Unity development engine, it can do so using its own fbx library without using any sort of API.
- **OBJ:** Some of the main reasons for using this file format are for its ability to handle high resolutions without increasing the file size and it’s a simple and open file format, used by a wide variety of 3D software for export and import. This makes sharing 3D model as an Obj file great as most of the software out there will be able to interpret it correctly and consistently. Furthermore, an Obj file is going to be much more lightweight and small in file size compare to fbx of the same model.

- **MB:** The MB format is a binary scene file used with the Maya 3D software. It stands for Maya Binary scene, the file contains 3D models, textures, lighting and animation data that can all be used with the Maya 3D software.
- **TGA:** Truevision Graphics Adapter Image file, or also known as Targa Graphic file, Truevision TGA or TARGA. The tga format is often associated with the video games industry and for texture files used with 3D models. It supports 8,16, 24 or 32 bits per pixel at a maximum of 24 bits for RGB colours and 8-bit alpha channel.
- **PNG:** The png or the Portable Network Graphic is an image storage format. It uses lossless compression and contains a bitmap of indexed colours. It was originally created as a solution to limitations of the GIF format, mainly to increase the colour support and to provide a format without the patent license. The format also includes the ability to use an 8-bit transparency channel, allowing the fading from opaque to transparent.

The average poly count of the scene at one time is around 2 million for tris and 6 million for the verts. The draw calls or batches, as named in unity, are about an average of 1500 for the scene at one time.

THIRD PARTY TOOLS

The third party tools that we will be using for the project are the following:

- **Unity 2017.3.0f3:** The game engine. https://unity3d.com/
- **Sourcetree 2.6.9.0:** The main version control we are going to be using. https://www.sourcetreeapp.com/
- **Visual Studio 2015/2017:** The main IDE we will be using for writing gameobject scripts. https://visualstudio.microsoft.com/
- **Google Chrome:** For basic research and unity documentation. https://www.google.com/chrome/
- **Google Drive:** For the storage of all development documentation, builds and art asset. https://www.google.com.au/drive/
- **Trello:** Online management software for team collaboration and to-do lists. https://trello.com
- **Draw.io:** Graphing software for creating development documentation. https://www.draw.io/
- **Discord:** Communication software for organizing work outside of work hours, sick days and other basic project communication. https://discordapp.com

- **GitHub:** Repository service for the use with our source control software. https://github.com
- **Cinemachine 2.1.10:** Camera software for cutscenes and the player controller. http://www.cinemachineimagery.com/
- **ProCore:** Used for block outs with the pre-development versions of the project. http://www.procore3d.com/
- **OBJExporter:** For the use with exporting unity scenes into wavefront obj file formats. https://assetstore.unity.com/packages/tools/utilities/scene-obj-exporter-22250
- **xInput:** API for managing connected controllers with unity. https://github.com/speps/XInputDotNet

These tools will help with the creation of our project to the best possible standard.

SYSTEM REQUIREMENTS

The Minimum System Requirements for this project are the following:

- Operating System: Windows XP SP2+, Mac OS X 10.9+, Ubuntu 12.04+.
- Graphics Card: DX9 (Shader model 3.0) or DX11 with feature level 9.3 capabilities.
- CPU: SSE2 instruction set support.

These are the requirements that are listed on Unity's official website for running a unity game.

Link to the official unity requirements page:
https://unity3d.com/unity/system-requirements

GAME SYSTEMS

OBJECTS, SCRIPTS AND SYSTEMS

Player:

Description: A quadruped dog, the main character of the game controlled by the user. Goes forward, backwards, left, right and jumps from controller input as well as other various things.

Script: The main function of the script will be updating the player transform using xInput (Third party tool for using controllers), basic camera control and jumping. The player will move forward, backwards, left and right relative to the camera. For example, regardless of camera orientation, the player will always move left if holding the left button, if the user also rotates the camera left during this it will cause the player to rotate in a constant circle. Other functions that will take place in this script are the interactions with objects, this will be done by flicking a static bool value off and then on again so it can be accessed by the scripts of other objects that will require interaction.

Exposed Variables:

- Walking Speed.
- Running Speed.
- Jump Force.
- Rotation Speed.
- Main Camera.

Day Manager:

Description: This object will manage the day-night cycle of the game, as well as what objectives can be completed on each day.

Script: This script will mainly be used for timers for the day cycle, switching the day, dimming lights, changing the skybox, and other day cycle

functions. Another important function, however, will be enabling different objectives depending on the day.

Exposed Variables:

- Day Length.
- Amount of Days.

Game Manager:

Description: The Game manager will manage the pause state, objective progression, background music, UI pop-ups, and other background game logic.

Script: This script will have an objective complete function that will be called by an objective script telling it which objective has been completed, this will mark it as compete for the todo list UI. This script will also handle the different UI pop-ups that will occur throughout the game, for example, the todo list. The todo list will pop up when an objective is complete as well as when the todo list button is pressed on the controller, this will be the same for the pause screen. Playing and changing any background music or sound effects will also be done in the game manager.

UI:

Description: This will be multiple different scripts depending on the UI used throughout the game, these scripts will control how the UI looks and functions.

Script: UI scripts will be basic and consist of basic logic and mainly visual altering code.

Exposed Variables:

- Colors.
- Text.
- Size.

Hot dog Wrapper:

Description: Simply an object that triggers the hot dog wrapper objective when interacted with. Creates a scent trail leading to the hot dog object.

Script: This script will be responsible for starting the first hot dog objective by interacting with the hot dog wrapper. Once the interaction is complete the script will start spawning all of the scent particles to create a path to the hot dog stand for the player to steal. Once the player is successfully lead to the hot dog stand the particles will be despawned.

Exposed Variables:

- Path Starting Position.
- Path Ending Position.
- Path Color.
- Path Spacing.

Hot dog:

Description: Simply an object that completes an objective when interacted with. The hot dog objectives will be marked as complete on the todo list upon interaction.

Script: This script will check the interaction between the hot dog object and the player, if the interaction is successful then the current objective will be checked and then mark as complete.

Music mini-game:

Description: An empty gameobject attached to an object that requires the music mini-game. Object contains the logic of the mini-game.

Script: This script will contain the logic of a music rhythm mini-game. This game will function by checking button presses in conjunction with timed animations on screen, if the timing is correct you'll get a point and eventually complete the objective.

Frisbee:

Description: Simply an object that completes an objective when interacted with. The frisbee objective will be marked as complete on the todo list upon interaction.

Script: The frisbee object will simply be a lerp on a loop, like the hot dog, this script will check the interaction between the frisbee object and the player, if the interaction is successful then the current objective will be checked and then mark as complete.

Exposed Variables:

- Lerp Speed.
- Lerp Arch.
- Lerp Starting Point.
- Lerp Ending Point.

AI Seek:

Description: An AI seek behaviour used for several NPC, the boys that chase you after stealing their frisbee and the hot dog owner on the third day. The hot dog owner will go slower if wet.

Script: This will function as a player seeking AI script, will be reused for a few different NPC types and will have public values for speed and other seek customisation.

Exposed Variables:

- Seek Speed.
- Seek Timeout.

Bin \ Sprinkler:

Description: Bin \ sprinkler object, when interacted with will knock over or turn on.

Script: The main function of the script will be playing animations on player interaction. This script will be used for a few different interactable objects and so will require public values for specifying the object type.

Exposed Variables:

- Enumerator for Object type.

Bin \ Sprinkler Manager:

Description: This manager will keep track of the knocked over bins or sprinklers and activate the next element of the objective once a certain amount is met.

Script: Will manage how many bins or sprinklers have been interacted with by the player, once the interactions hit a specified number the objective will be marked as complete for that specific object type.

Exposed Variables:

- Interaction Amount.

Hot dog stand Owner:

Description: Simple AI object that will walk from the hot dog stand and pick up each of the knocked over bins from the player.

Script: This will function similar to an AI seek behaviour, however, will seek towards specified locations. Once the hot dog stand Owner arrives at a location it will play an animation and then continue the seek to the next location after that animation is complete.

Exposed Variables:

- Locations Amount.
- Array of Locations (Location 1, 2, etc).
- Seek Speed.

Butterfly:

Description: Butterfly object will simply disappear when interacted with, or activate an objective if none have been interacted with yet.

Script: A simple script that checks interaction from the player and object, animates and turns off the object on success. If a butterfly is marked as an initial butterfly then it will start spawned and will act as the activation object for the Butterfly objective.

Exposed Variables:

- Initial Butterfly.

Butterfly Manager:

Description: The butterfly manager will keep track of how many butterflies have been interacted with, how much time is remaining and mark an objective as complete.

Script: The main function of this script will be managing how many butterflies have been interacted with, counting the timer and marking the objective as complete when the interacted value equals the Butterfly amount value. If the timer finishes and not all butterflies have been interacted with than any remaining butterfly will be unspawned and objective cannot be complete.

Exposed Variables:

- Timer amount.
- Butterfly amount.

Photobomb:

Description: An empty gameobject attached to some animated objects taking a photo. Will complete an objective when interacted with at the correct timing.

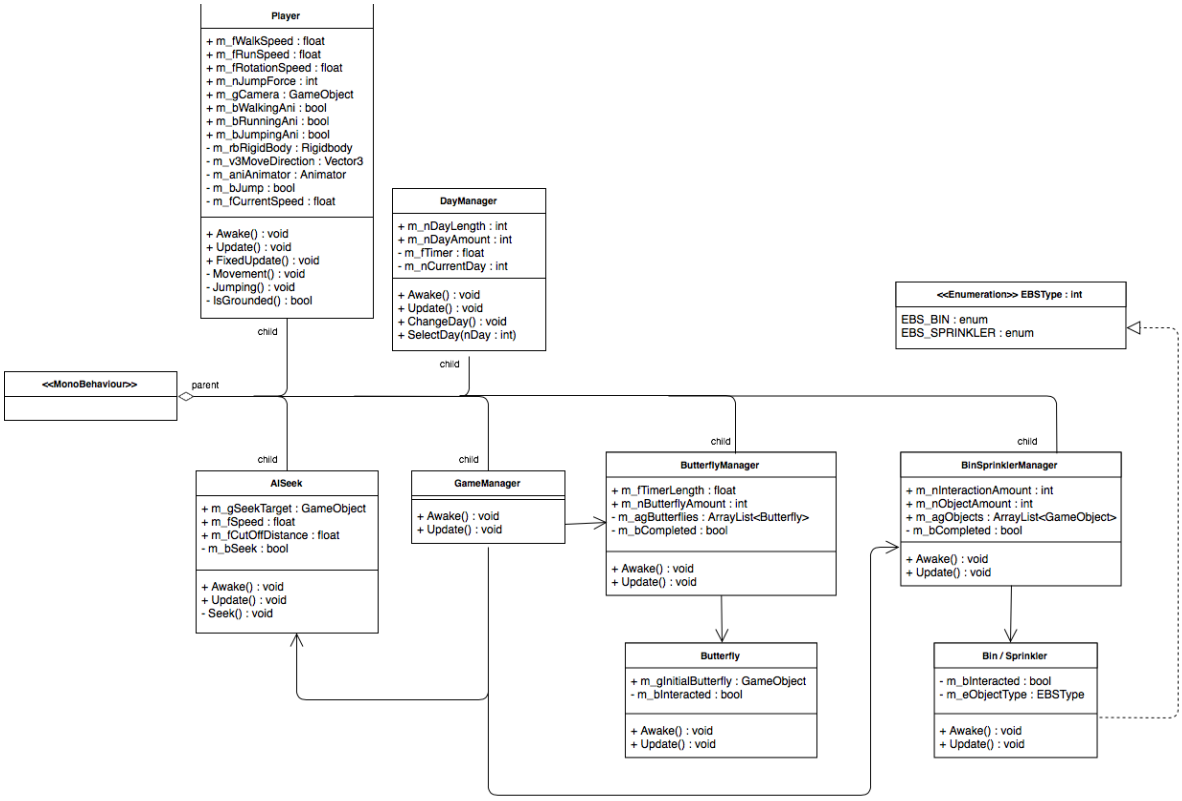
Script: The script will be a simple check for interaction at the correct time of an animation. A timer will start and reset with the animation, if the interaction from the player happens within a set time of this timer then the objective will be marked as complete.

Exposed Variables:

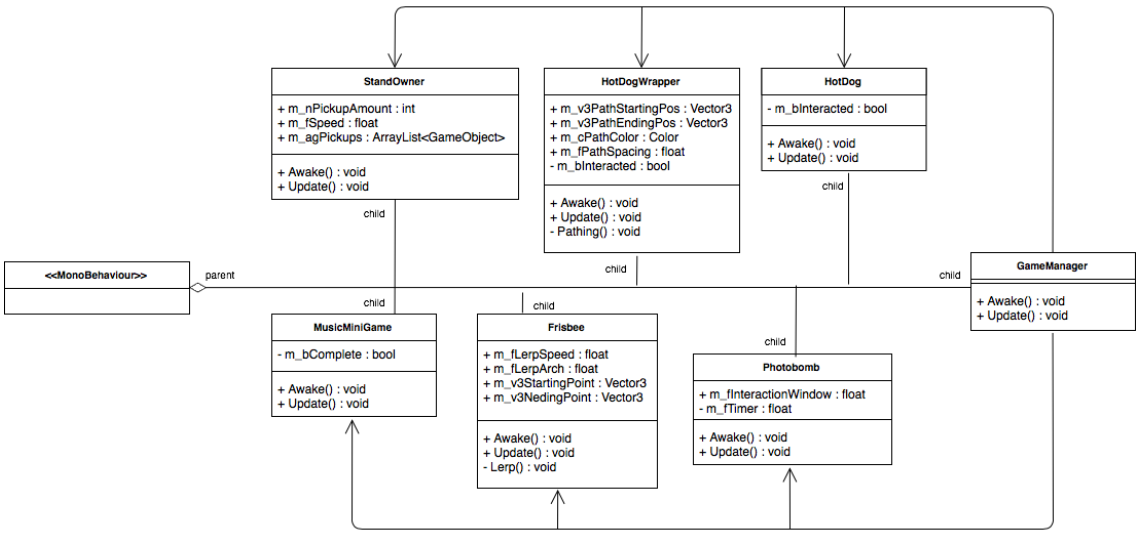
- Interaction window.

CLASSES

This is a class diagram of some of the main classes in the system:



This is a class diagram of some of the objective / mini games in the system:



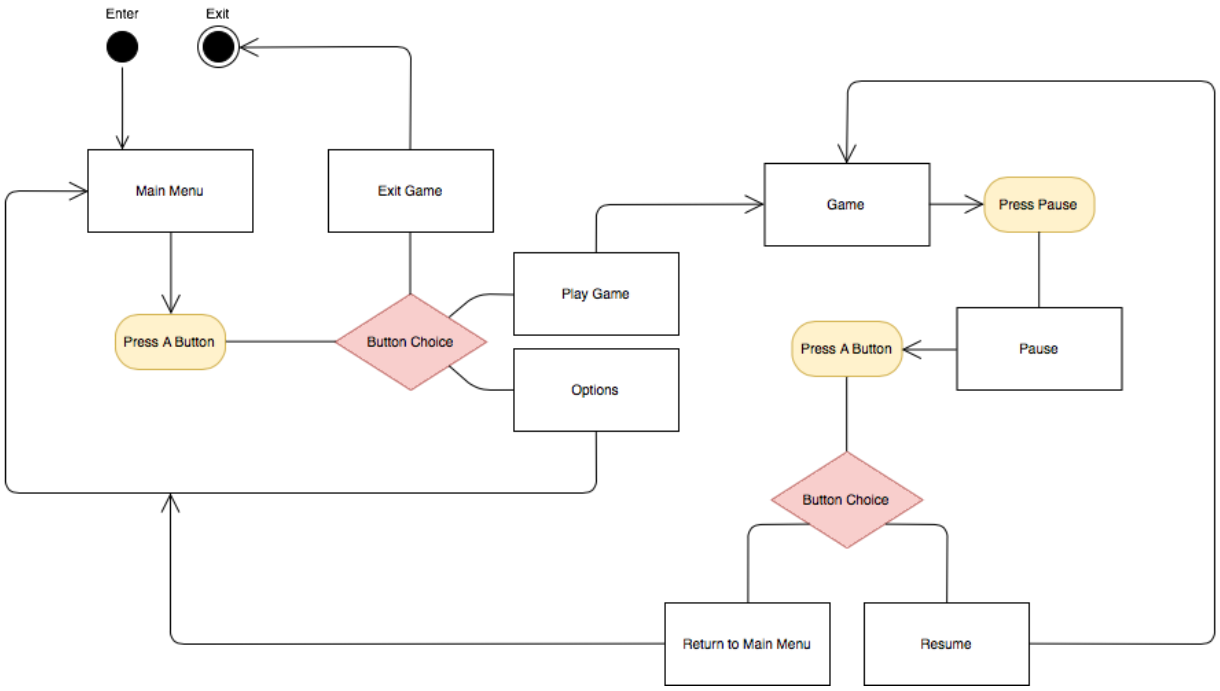
USER INTERFACE

MENUS AND GAME UI

There will be 3 main areas of UI throughout the project, the main menu, the to-do list and the pause menu. Other than these 3 main areas of UI the game plans on being relatively UI free.

Menus:

- **Main Menu:** Main menu will consist of 3 buttons for ‘Play’, ‘Credits’ and ‘Exit’.
- **Pause Menu:** Similar to the main menu but this menu will consist of ‘Resume’, ‘Main Menu’ and ‘Exit’ instead.



Game UI:

- **To-do List:** A simple slide in and out text list of available player objectives. Once an objective is complete it will be automatically marked off of this UI list.

- **Bone Collected:** Once one of the bone collectables scattered throughout the game is collected a UI element showing how many have been collected (and how many left to collect) will appear in one of the bottom corners of the screen.

Implementation of UI:

- **Menus:** Main Menu and Pause menu will be implemented using the Unity canvas system. Menu buttons will be selected with the left analog stick, A button to confirm the selection, and B button to go back if available.
- **To-do List / Bone Collected:** The to-do list and Bone collected UI will also be implemented using the Unity canvas system. The list is purely visual with simple text, no interactive elements other than bringing it to the screen, and ticking / removing completed objectives, the Bone Collected will be the same.

INPUT METHOD

The Input Method for the game will be through one connected controller for one player. A number of buttons will be used for different player abilities, including movement, jumping, interacting, camera movement, and other player related controls.

We will be developing and testing with Xbox 360 and Xbox One controllers, other controllers, even keyboards, will work with our game, however, the intended controller to be used during development and after will be the Xbox controller.

Gameplay:

- **Left Analog Stick:** Move the player forward, back, left and right.
- **Right Analog Stick:** Rotate and zoom the player camera.
- **Y Button:** Dog Howl, simply plays a piece of audio.
- **A Button:** Jump the player. Quick presses are smaller jumps and holds are bigger jumps.
- **B Button:** The interact button, used to interact with objectives and activations.

- **Left Bumper:** Sprint button, press and hold to increase the player movement speed.
- **Back Button:** Bring up the players to-do list of available objectives.
- **Start Button:** Pause button, brings up a menu and pauses the game.

CODE OVERVIEW

CODING CONVENTIONS

Naming Convention:

In this project, we will be writing functions and classes in Camel case (eg. ThisFunction and ThisClass) and variables we will be using Hungarian Notation. If a variable is a member we will be specifying this using a “m_” followed by a single letter representing the value type (eg. m_fMyVar). A full chart of all the different notation is in the table below.

Hungarian Notation Guide:

Standard Notation	Meaning	Example
m_	Member variable.	Player m_Player
s_	Static variable.	static float ms_fTimer / static float s_fTimer
n	Integer type.	int nValue
u	Unsigned integer type.	unsigned int uValue
b	Bool type.	bool bAlive
f	Float type.	float fTimer
c	Char type.	char cLetter
v3	Vector 3.	Vector3 v3Position
m4	Matrix 4.	Matrix4 m4Transform
a	Array.	Player aPlayers[10]
E and e	Enum type. E for declaration, e for variables of type.	EWeapon / EWEP_RPG / eWeapon

Unity Specific Notation	Meaning	Example
g	Gameobject type.	Gameobject gEnemy
rb	Rigidbody type.	Rigidbody rbBody
as	Audio Source type	AudioSource asAudioSource
ac	Audio Clip type.	AudioClip acDeathSound
c	Color type.	Color cCarColor

mat	Material type.	Material matCat
m	Mesh type.	Mesh mCatMesh
s	Script type.	PlayerScript sPlayer

COMMENTING

General code:

```
// Call the function pointer.
int Callfunction(AStarNode* pStart, AStarNode* pEnd);
```

Throughout the project, we should be using single line comments above code lines as much as humanly possible to elaborate code flow.

Functions:

```
//-----------------------------------------------------------------------------
// CalculatePath: Calculate the Astar path.
//
// Returns:
//      bool: Returns true or false if a path is found or not.
// Param:
//      pStart: AStarNode pointer for the start of the path.
//      pEnd: AStarNode pointer for the end of the path.
//      finishedPath: A DynamicArray pointer of AstarNode pointers.
//-----------------------------------------------------------------------------
bool CalculatePath(AStarNode* pStart, AStarNode* pEnd, DynamicArray<AStarNode*>* finishedPath);
```

When commenting functions we should open the comment with some sort of comment line, then when closing do the same. Between the lines, we should have a title which has the name of the function and what it is doing. Under the title should be what parameters are taken in, if there is any as well as what is being returned if there is something being returned.

Classes:

```
//-----------------------------------------------------------------------------
// AStarNode object. A Star pathfinding algorithm.
//-----------------------------------------------------------------------------
class AStar
```

Classes should be commented similar to the way functions are within the comment lines, and in between the lines having the class name, what it inherits from and what it does.

Scripts:

```
//--------------------------------------------------------------------------------
// Purpose: Player Functionality.
//
// Description: The Player script is gonna be used for controlling each player when it is
// their turn. This script is to be attached to an empty gameobject for each player.
//
// Author: Thomas Wiltshire
// Edited By: Jeff Jeffer
//--------------------------------------------------------------------------------
```

All scripts are to be topped with a comment slab like the example above. This will be used to explain the purpose of the script, but also acting as a credit area for the author of the script and also anyone that has edited it.

Marker Tags:

- **TO-DO**: We will use this comment tag at the start and end of code blocks that are incomplete or need to be completed.
- **RECODE**: We will use this comment tag at the start and end of code blocks, it will be used to mark if something needs to be recoded.

UNITY ATTRIBUTES

All public values throughout the project will use the unity provided attributes to organized the gameobject's inspector. These will help designers with how to use public values, restrict how values are used, and change variable names so they are more understandable.

- **[Header("Player:")]:** Used to separate public values into sections with headings to specify value use.
- **[LabelOverride("Player Number")]:** Overrides public value names from code to something more relevant to designers.
- **[Range(1,2)]:** Used for specifying a range between two numbers for public number values. Useful for making sure the correct numbers are chosen.
- **[Tooltip("Which player is this?")]:** Tooltips for helpful tips on how to use the public value, descriptions or other helpful notes.
- **[Space]:** Simply used to space out public values, handy for use with the [Header] tag for separating public values.

- **[HideInInspector]:** Mark a public value as hidden from the unity inspector. Used for when values need to be public but don't want to be altered by designers.

CODING GUIDELINES

- Keep up to date with commenting code.
- Careful with variable naming, always make sure they are clear, clean and make sense.
- Make sure to use the Marker tags if something is incomplete, needs to be recoded, etc.
- All variables are to have private or public written in front of them, regardless of default access type. This is for better readability and consistency.
- Getters and Setters, if required, are to be at the top of the script under variable declaration.

SOURCE CONTROL

The unity project will be kept on a central GitHub repository using Sourcetree to check in and check out each day.

To help us keep organized and error-free here are some things to remember while using source control:

- All code that is checked into the repository MUST compile and without error, as to make sure not to mess up other people's work or of course the project.
- Always make sure that any unnecessary files (eg. visual studio project files) are not uploaded to the repository and are properly ignored using an appropriately set up gitignore file.
- Make sure that commits have clear and well-formatted titles, describing what is being checked into the repository. If more information is needed in a commit, create a short brief title with a longer description to describe the changes.

PROJECT TEAM

MEMBERS

Here is a list of team members on the project. Programmers at the top with their name and what they are doing on the project next to it, the rest of the team will be under a title with their name.

Programmers:

- Thomas “Jeff” Wiltshire - Player controller, Story events, other basic programming tasks.

Designers:

- Mickey Seamark.
- Emma Wearing.

Producer:

- Zane Talbot.

Artist:

- John Hickton-Burnett.
- Ee Ching Lai.
- Samuel Rau.

TEAM SIGN-OFF

Designers:

Artists:

Programmers:
