Toy Story: The Escape Plan

Game Design Document (GDD)

Game Programming Course

Professor:
Carmelo Macrì

Students:
Armando Pezzimenti (ID 204895)
Davide Amato (ID 199670)

Department of Mathematics and Computer Science, UNICAL

UNIVERSITÀ DELLA CALABRIA DIPARTIMENTO DI MATEMATICA E INFORMATICA

Made with unity

Academic Year 2020-2021

Contents

Game Overview

1.1 General Information

- **Number of players**: Single player;
- **Genre**: platform, puzzle, adventure;
- **Perspective type**: Third person;
- **Projection type**: Perspective;
- **Platform**: PC;
- **Tools used to play**: Mouse & keyboard;
- **Tools used to develop the game**
  - **Game core**: Unity (2019.3.7f1), Microsoft Visual Studio 2019;
  - **Comunication and remote pair programming**: TeamViewer.
  - **Graphics**: Gimp, Procreate (IPad).
  - **Audio**: Audacity.
  - **Models**: Blender.
  - **Verison control system**: Git & GitHub (*Repository link here*)

1.2 Introduction

Toy Story: The Escape Plan is a project fully inspired by the **Toy Story** © franchise. It has been made as final project for the Game Programming exam at the University of Calabria. It's a platform/puzzle game for PC where a toy (main character) wants to escape from a house full of obstacles and characters to interact or confront with.

1.3 Look & Feel and Target Audience

The game tries to recreate the style of the **Toy Story** © franchise, giving that kind of **look & feel** that brings us back to our childhood, with a cartoon style, using low poly graphics, saturated colors, funny VFXs and SFXs. For what concern the **target audience**, this game aims to reach not only children, but also adults, giving puzzles and fun for every age.

1.4 External influences

1.4.1 Games

Toy Story 2

COD4: Rat Houde Mod Map

Kingdom Hearts III: Toy Box

1.4.2 Films

Toy Story Saga

Small Soldiers

2.1 Story Telling

It has been a long time since Andy stayed home playing with his favorite toys. He became an adult and, like all adults, he forgot to see his toys as friends.
Woody and all the other toys have been forgotten, used only as ornaments. The player's goal is to help Woody to escape from the room so that he can become an important playmate for some other child.
Inside the room there are some objects that will help the player to improve his skills in order to achieve some goals including:

- collecting other toys: all of Woody's friends are inside the room, but they are now inanimate, the imagination has abandoned them;
- interaction with objects, including aliens and Andy's beloved console that will also provide additional suggestions for the completion of Woody's mission;
- escape from Andy's mother that is now paying attention to every single noise, because she has forgotten what it means to have children playing at home.

The game ends with Woody walking out the main door with all his friends.

2.2 Characters

2.2.1 Woody

Woody is the main character of the game. Woody is one of the most important characters in Toy Story saga. All the actions that the player performs during the game will affect Woody's journey through the level.

2.2.2 Jessie

Jessie is one of the collectibles. She is one of the toys that was not part of the first episode of the saga but, once she joined the group, she never left proving to be important for the other toys.Woody has to find her so he can get out of Andy's room.

2.2.3 Bullseye

Bullseye is one of the collectibles. He is one of the toys that was not part of the first episode of the saga but once he joined the group, he proved to be a great friend.Woody has to find him so he can get out of Andy's room.

2.2.4 Rex

Rex is one of the collectibles. One of his strengths is certainly his sympathy, in addition to the fact that he is a bit clumsy. Woody has to find him so he can get out of Andy's room.

2.2.5 Slinky Dog

Slinky Dog is one of the collectibles. He is one of Woody's most loyal toys, he would never have wanted Woody to leave the room without him. Woody has to find him so he can get out of Andy's room.

2.2.6 Buzz Lightyear

Buzz Lightyear is one of the collectibles. Despite Woody's initial dislike, he proved to be a great friend and mate for him. Woody has to find him so he can get out of Andy's room.

2.2.7 Hamm

Ham is one of the collectibles. He is a piggy bank. Woody has to find him so he can get out of Andy's room.

2.2.8 RC

RC is a radio-controlled car, one of Woody's friends in the Toy Story saga. Woody has to find him so he can get out of Andy's room.

2.2.9 Andy's mom

Andy's mother is the only one who can stop Woody's journey. He needs to escape from her so he can get out of the room and save his friends.

2.2.10 Aliens

Aliens are interactible objects. Woody has to listen to their suggestions to improve his skills and get out of Andy's room.

2.2.11 PSOne

The PSOne is Andy's console. It is an interactible object. Woody has to be aware of its suggestions to know how to escape Andy's mom.

3.1 Introduction

In this game the character used by the player is Woody. The game starts with Woody waking up inside Andy's room. The player will have to find a way to get out of Andy's room. Woody will have to open the door but, being too small, he will have to attract someone to open the door in his place. Andy's mother, Jennifer, being always in the house, is the only one who can do it. The player must be very careful that Woody is not seen alive by humans, because they may be frightened to see a doll "*alive*"(it will be better explained later). In order to escape from the house, Woody must overcome obstacles, defeat enemies, unlock abilities and characters, solve puzzles and many other activities. Once out of the front door, Woody can finally go looking for a house where there is a child who still has the pleasure to play with him.

Demo Because of time restrictions, the work is not fully finished. It's just a Demo presentation of the idea behind the project. You can find further information about the whole idea in chapter 8

3.2 Controls

The player can perform different actions by using the controls indicated below:

Foot Controls	
KEY	**ACTION**
W	forward
A	backwards
S	left
D	right
SPACE BAR	jump
LEFT SHIFT	run
MOUSE	look
MOUSE WHEEL	zoom in/out
F	interact
X	activate/deactivate ragdoll mode
C	collect items
ENTER	confirm

Hanging Controls	
KEY	**ACTION**
S	left
D	right
SPACE BAR	climb
Q	fall

3.3 Mechanics

3.3.1 World Physics

The **physical** universe of the game works pretty much like in the real world, with *gravity*, *kinematics*, etc.... **Not every object in the game is ruled by physics.** Some objects are *static*, others are *controlled via script*.

3.3.2 Movements and Actions

The **player** is able to perform the following **movements** and **actions**.

- **walk** and **run** with a range of motion of 360 degrees;
- **jump** both from stationary position and during walk or run movements. If the player jumps from a standstill, he cannot do movements in the air;
- possibility to **switch among 3 different modes**:

  Ragdoll Player can switch to ***ragdoll*** mode only when he's in *alive* mode. This mode is very useful to not being detected from Jennifer when there are no hiding spots nearby.

Alive Player can switch to ***alive*** mode both from *ragdoll* and *hidden* mode. To come back from *ragdoll* to *alive* mode, it's required that the player is almost still.

Hidden Player can switch to ***hidden*** mode only when he's in *alive* mode. This mode is quite useful for not being detected from Jennifer, while keeping the freedom to make small movements. To switch to *hidden* mode, it is required that the player is in proximity of some ***hideout*** object. When the player is close enough to some object, to figure out if it's an *hideout* object or not, he can target at it with the white dot in the middle of the screen. If the object is a *hideout* object, it will be outlined with a red line. Moreover, when the player is inside a *hideout*, he can do small and slow movements.

- **interact** with some objects and characters in the room;
- **hang** to some surfaces. When *hanged*, the player can **move left/right**, **fall** from the surface or he can **climb** it.
- **collect** some objects;

3.3.3 Interactable Objects

Every **Interactable object** will be **outlined** whenever the player point it with the "*pointer*" (the *white dot* positioned in the middle of the screen. Depending on the type of the *Interactable object* pointed, the *outline* will be of *different color*.

- **hideouts** (*red outline*): objects where the player can *hide* himself.
- **buttons like** (*blue outline*): the player can *switch on/off* these kind of objects. Some of these objects are *one-time use*
- **friendly characters** (*yellow outline*): the player can *interact* with this characters to *speak* with them.
- **collectibles** (*green outline*): the player can *collect* this type of object. Sometimes, the player can also ***interact*** with them in other ways too

Aliens

Aliens are the *mascot* of this game. Their aim is to help the user, guiding him during his journey. The **aliens' behaviour** is very simple:

- Initially they are ***still***, waiting.
- Whenever the player is getting close to them, they start to ***say hello*** in order to draw user's attention.

- Then, when the player is close enough and *interact* with them, aliens start to ***speak*** with the player in order to help him by giving some useful information.
- After that, when the interaction is finished, and the player goes far away from them, they ***come back to stay still***.

Since player can *interact* with Aliens, they are ***interactible objects*** and, in particular, they are part of the ***Firendly Characters***.

Other Characters

In Andy's room there are other recognizable characters. These characters are: **Buzz Lightyear, Jessie, Rex, Hamm, Slinky Dog, Bullseye** and **RC Car**.
The player is able to *interact* with them, making them emit some sound. Player can also *collect* them. In fact, they are part of the ***collectibles***.

Box

In the room there is a single **Box**. It can be used by the player to hide Woody inside. It is easy to understand that the Box is part of the ***hideouts***.

PlayStation One

Near the TV station there is an old-fashioned **PlayStation One**. The player will be very happy to know that he can interact with it, but just one time. Actually, he can **switch it on**. This makes PS1 part of the ***buttons like*** objects.

3.3.4 Mom

Andy's Mom is the only human in the house. Whenever she **hears** some **loud noise**, she is **alerted**, and goes to investigate where the noise comes from, looking around.
Moreover, she is provided with a **Field Of View**. If the player is inside her *FOV*, she's able to detect him, but only if the player is in *alive* mode.

3.4 Victory and Game Over conditions

In game goals Woody must distract Jennifer (Andy's mom) interacting with some object in the room. In order to escape from a room he must solve some puzzle to unlock the door.

End goal (victory) Woody escapes from the room.

Loss avoidance (Game Over) Woody must avoid to being seen alive from Jennifer. In other words, he must avoid to be inside the *Mom's FOV* while he is in *alive* mode.

4.1 Andy's room

This level, as already introduced, should be the demo level. So, in this level the player must learn what are the basic skills to use within the game. To make the game consistent with the Toy Story saga the best solution is to place the initial level inside Andy's room.
The objects inside the room are arranged consistently in order to reflect Andy's room in the Toy Story saga (Figure 4.1).

(a) Andy's room in the Toy Story saga.

(b) Andy's room built in Unity.

Figure 4.1: Comparison of Andy's room in the Toy Story saga with the one built in Unity.

4.1.1 Level Flow

The level consists of four main phases:

- Level introduction phase;
- Objects collection phase;
- Unlock mom routine phase;
- Escape from the room - final phase;

Level introduction

This phase is divided into two other phases:

- Room tour: in this phase a tour of the room is made using a camera in order to give the player a complete view of where the level takes place.
- Tutorial phase: in this phase the player has a first interaction with an Alien that will show him through a dialog what are the basic skills he must use at the beginning of the game. In particular the phrases that this alien says are:
  1. "OOOOOOOOOHHHHHHH. Woooodyyyyyyyy!!!!"
  2. "Happy to see you again. We were afraid you wouldn't wake up anymore."
  3. "Andy has grown and forgotten us."
  4. "Now you have a mission..."
  5. "You must bring us all out of this place. We must find a child to keep happy",
  6. "First of all, since you didn't move for so long, I'll help you a bit.",
  7. "Press W,A,S,D to walk!",
  8. "Press SHIFT to run!",
  9. "Use the MOUSE to look around and MOUSE SCROLL to zoom!",
  10. "Press X in order to appear dead. Press again X to enjoy the journey.",
  11. "You can interact with outlined objects by pressing F",
  12. "You'll find my brothers in your path that will help you. And remember...",
  13. "You got troubles, and I got 'em too. There isn't anything I wouldn't do for you...""

Collecting objects

At this stage Woody can perform several actions in order to find his friends. These actions are:

- interact with aliens, so that the player understands what are the other skills of Woody. In particular there are two aliens that Woody can interact with:
  - Alien for the hideout tutorial (Figure 4.2, ID 7): Once Woody interacts with this alien, he will tell to the player how to use the hideout skill of Woody using the box behind him. In particular the phrase that this alien says are:
    1. "Press F to hide inside the box! Once inside, press F again to exit."
  - Alien for the jump and climbing tutorial (Figure 4.2, ID 8) : Once Woody interacts with this alien, he will tell to the player how to use the jumping and climbing skills of Woody. In particular the phrases that this alien says are:
    1. "Press SPACE to jump!"
    2. "If you jump from still, you won't be able to move in air.",
    3. "Otherwise, you can change your direction in air.",

4. "Moreover, if you jump on some edges, you can hang on it.",
5. "Once hanged, you can:
   (a) move right/left.
   (b) you can fall down by pressing Q.
   (c) you can climb the surface by pressing SPACE."

- Alien for the base movement tutorial (Figure 4.2, ID 10): is the first alien introduced in the previous phase.

- collect other toys, in order to escape from the room Woody has to take all his friends with him. At this step the player must have all the necessary skills to collect all of Woody's friends. To test the player's skills the toys are strategically placed inside the room:

  - Buzz Lightyear (Figure 4.2, ID 1): is placed on a shelf inside the room, the player will have to make a path using some of Woody's skills such as jumping and climbing to reach him.
  - Slinky Dog (Figure 4.2, ID 2): is placed on the bedside table next to Andy's bed, the player must use some of Woody's skills to reach him.
  - Jessie (Figure 4.2, ID 3): is placed in one of the boxes left by Andy before he moved to the college. The player must find the correct way into the maze of boxes to reach her.
  - Rex (Figure 4.2, ID 4): is positioned on the wardrobe; the player must use some of Woody's skills including jumping and climbing to reach him.
  - Bullseye (Figure 4.2, ID 5): is placed on Andy's desk near the computer; the player must use some of Woody's skills.
  - Hamm (Figure 4.2, ID 6): is placed on one of the shelves in the library; the player must use some of Woody's skills to reach him.
  - Rc (Figure 4.2, ID 9): is placed on the floor near the video game station; the player must use some of Woody's skills to reach it.
  - PSOne (Figure 4.2, ID 11) is placed on the floor; the player must have collected all the collectables to be able to turn on the PSOne, otherwise a Woody dialog will be activated:
    * "Oh no...I can't now, maybe i need some help!"

Figure 4.2: Placement of collectibles and interactable objects in Andy's room.

Unlock mom routine

To reach this phase, as introduced, the player must interact with the PSOne once all the toys inside the room have been collected. Once the previously introduced condition occurs there are two phases:

- **PSOne opening phase**: the camera only shows the television that plays the PSOne's power-on screen, at the end of it a video starts showing a message on the screen that warns the player that Andy's mom has heard the noise and is about to arrive.
- **Count down, hiding phase**: The countdown begins and the camera returns to Woody, the player must now (through the skills he has learned) find a way not to let the mother of Andy come in and see Woody alive.
- **Mom entering in the room phase**: the door opens, mom enters and the player must be careful so that she doesn't see Woody alive. At this point the player should use ragdoll or hiding skills or use the objects in the room to hide.

Escape from the room

Once Andy's mom enters the room she performs a routine. If she sees Woody alive then the player will loose the game. If Woody will be able to leave the room and Andy's mom does not see him alive then the player will win the game.

4.2 Final Scene

In the final scene, the camera takes a tour of the room showing the characters dancing with music in the background and scrolling credits.

UI design

The UI is composed by 3 main parts:

- **Menu**
- **HUD**
- **Game Over screen**

5.1 Menu

There are 2 menus: **Main menu** and **Pause Menu**. They are almost the same, except for one button.

5.1.1 Main Menu

Figure 5.1: Main menu

It is shown up when the game is run. It's composed by 3 buttons:

- **Start**: the actual game starts
- **Settings**: the *Settings menu* is shown
- **Exit**: quit the game.

Figure 5.2: Main menu buttons

Settings

Figure 5.3: Settings screen

The **Settings** menu:

- contains the ***Controls*** panel, that the player can see to check the controls and the actions allowed;
- allows the user to customize the following parameters:
  - ***Music***:
    - **Menu** : possibility to *increase/decrease* the menu music's *volume*;
    - **In-game**: possibility to *increase/decrease* the in-game background music's *volume*;

Figure 5.4: Music settings

– ***VFX***:

  * **Full screen**: switch between *windowed* and *full screen* mode.

  * **Resolution**: change game's *resolution*.

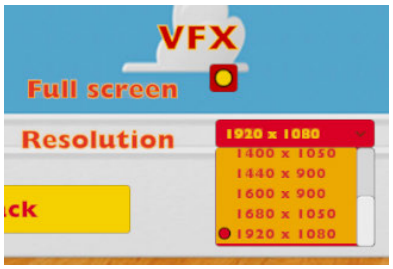


Figure 5.5: VFX settings

This menu presents just one button, the **Back** button, that the user can press in order to come back to the *Main/Pause menu*.

5.1.2 Pause Menu

This menu can be shown by pressing **ESC** during the game. It is exactly the same as the *Main menu*, except for the *Start* button that here is replaced by the *Resume* button, used for coming back to the game.

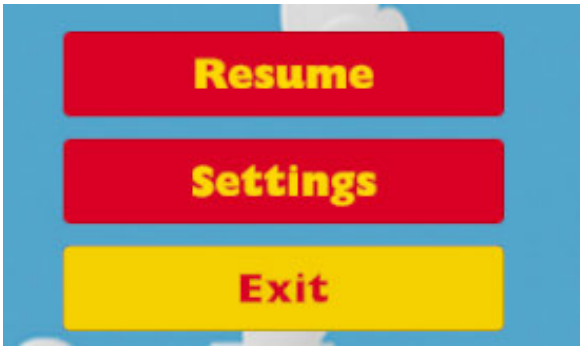


Figure 5.6: Pause menu buttons

5.2 HUD

The **HUD** is composed of 2 main parts:

- **Player UI**
- **Dialog boxes**

Player UI

The UI components that are part of **Player UI** are those elements that are always visible during the game. These components are:

- **Pointer**: it is the *white dot* positioned in the center of the screen. It can be used by the player to point the objects with which he wants to interact.

Figure 5.7: pointer

- **Collected items**: It is a simple list, positioned in the bottom-left corner of the screen, that lists an icon for each collected item.

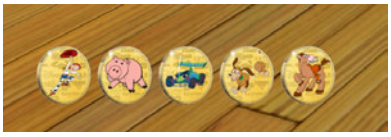


Figure 5.8: Collected items

Dialog boxes

In the game there are various dialogues that can be of 2 types: **alien dialogues**, that are shown when the player interact with aliens, and **player thoughts**, used to give some hints to the player.

(a) Alien's dialog

(b) Woody's thought

Figure 5.9: Examples of dialogues present in the game.

5.3 Game Over screen

Figure 5.10: Game Over screen

The **Game Over screen** shows ***a text*** that informs the player the *reason* why the game is over (giving some hints to avoid loosing again the next time), and has the ***exit button*** to quit the game.

5.4 UI flowchart

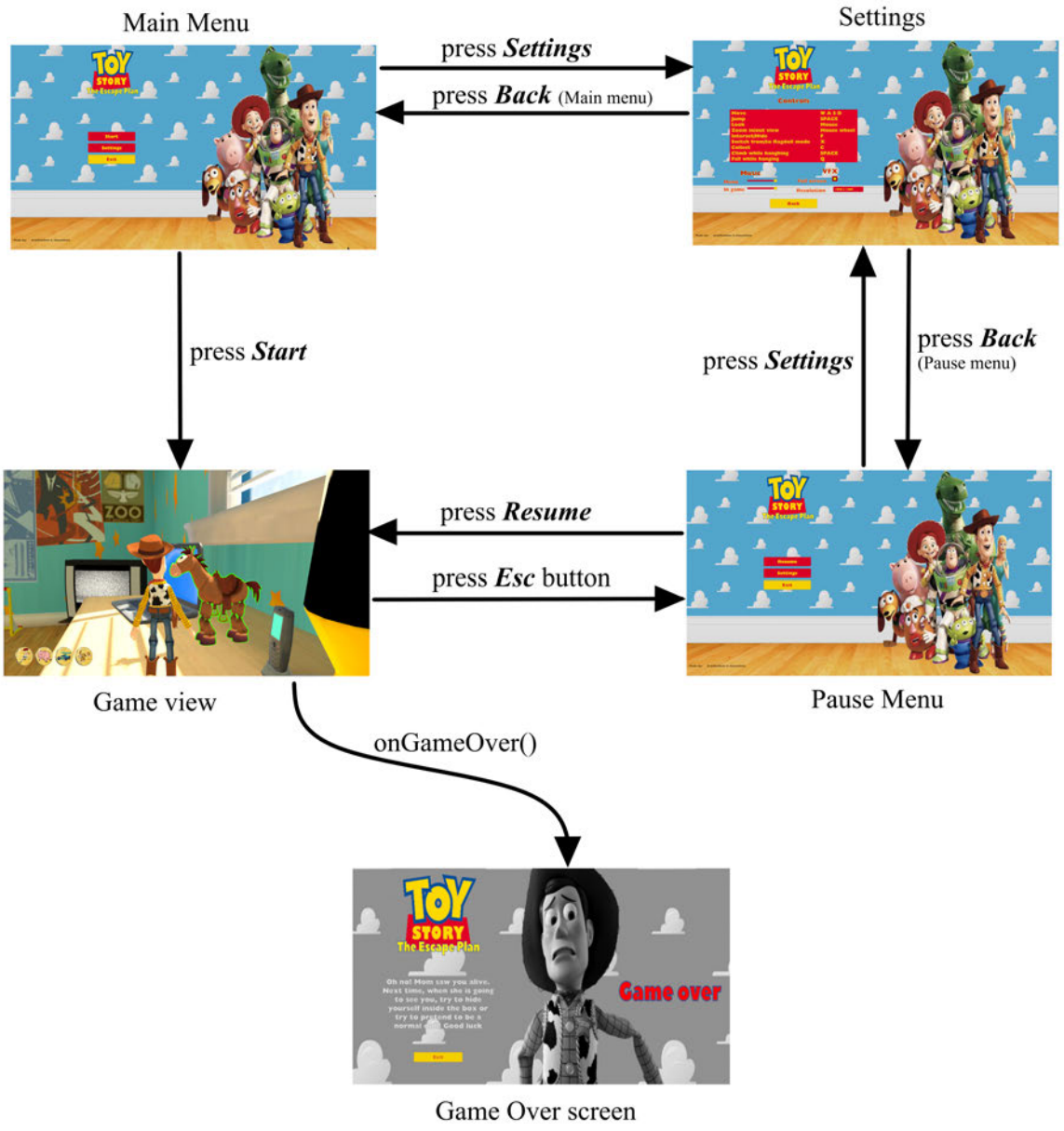


Figure 5.11: Image that represents the *UI flowchart*

Art Assets

6.1 3D models

The 3D models used are all in .fbx format, this is because, thanks to this format, we can also add animations to objects. Most of the objects used in the level are available for download at the link.
In addition, we used the famous tool called Blender that allowed us to modify some objects in order to adapt them to our project idea.

6.1.1 Characters

It is possible to differentiate the different 3D models into categories:

- Player: the 3D model used for the main character (Woody);
- NPC: the 3D models related to objects that are not controlled by the player;
- Andy's room: all 3D models that compose Andy's room;
- Collectibles: all 3D models related to the objects that can be collected by the player;
- Interactive objects: all 3D models related to the objects with which the main character can interact;

Player

The model concerning the main character, in this case Woody is a model in .fbx format. Furthermore this model is rigged, it has its own skeleton. The presence of the skeleton has allowed us to add animations to the character. The model is available at the link. The figure below (figure 6.4) shows the model in the usual T-Pose from 3 different perspectives.

(a) Woody's model in T-Pose front

(b) Woody's model in T-Pose side

(c) Woody's model in T-Pose back

Figure 6.4: Woody's 3D model in T-Pose from 3 different perspectives.

NPC

3D models related to characters that are not under the direct control of the player are:

- aliens: each of them uses a 3D model in .fbx format.Aliens are used for the part of the game related in learning Woody's skills. For this model we don't have a skeleton so we used the auto rigging tool provided by Mixamo on a model in .obj format so we can export the rigged model in .fbx format. The model is available at the link. The figure below (figure 6.8) shows the model in the usual T-Pose from 3 different perspectives.

(a) Alien's model in T-Pose front

(b) Alien's model in T-Pose side

(c) Alien's model in T-Pose back

Figure 6.8: Alien's 3D model in T-Pose from 3 different perspectives.

- Andy's mom: in this case we use a 3D model in .fbx format. As introduced Andy's mom is one of the NPCs and for this reason it is rigged so that we can associate animations to it. The model is available at the link. The figure below (figure 6.12) shows the model in the usual T-Pose from 3 different perspectives.

(a) Andy's mom model in T-Pose front

(b) Andy's mom model in T-Pose side

(c) Andy's mom model in T-Pose back

Figure 6.12: Andy's mom 3D model in T-Pose from 3 different perspectives.

Collectibles

The collectible objects to which the 3D models are related are:

- Buzz Lightyear: in this case we use a 3D model in .fbx format. The model is available at the link. The figure below (figure 6.16) shows the model in the usual T-Pose from 3 different perspectives.

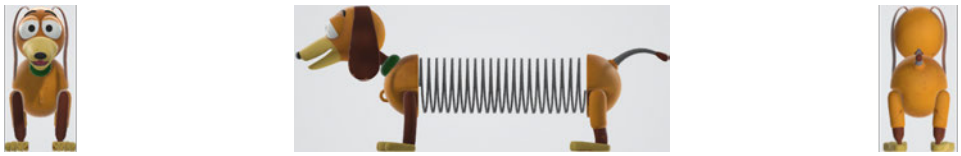
(a) Buzz Lightyear model in T-Pose front

(b) Buzz Lightyear model in T-Pose side

(c) Buzz Lightyear model in T-Pose back

Figure 6.16: Buzz Lightyear 3D model in T-Pose from 3 different perspectives.

- Slinky Dog: in this case we use a 3D model in .fbx format. The model is available at the link. The figure below (figure 6.20) shows the model from 3 different perspectives.



(a) Slinky Dog model front (b) Slinky Dog model side (c) Slinky Dog model back

Figure 6.20: Slinky Dog 3D model from 3 different perspectives.

- Rex: in this case we use a 3D model in .fbx format. The model is available at the link. The figure below (figure 6.24) shows the model from 3 different perspectives.

(a) Rex model front (b) Rex model side (c) Rex model back

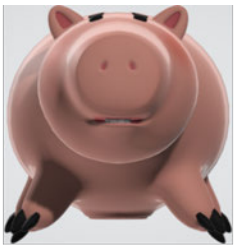
Figure 6.24: Rex 3D model from 3 different perspectives.

- Bullseye: in this case we use a 3D model in .fbx format. The model is available at the link. The figure below (figure 6.28) shows the model from 3 different perspectives.

(a) Bullseye model front (b) Bullseye model side (c) Bullseye model back

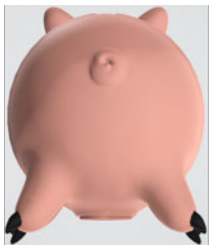
Figure 6.28: Bullseye 3D model from 3 different perspectives.

- Hamm: in this case we use a 3D model in .fbx format. The model is available at the link. The figure below (figure 6.32) shows the model in the usual T-Pose from 3 different perspectives.



(a) Hamm model front

(b) Hamm model side

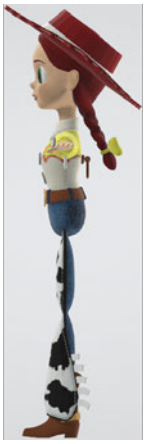


(c) Hamm model back

Figure 6.32: Hamm 3D model from 3 different perspectives.

- Jessie: in this case we use a 3D model in .fbx format. The model is available at the the link. The figure below (figure 6.36) shows the model in the usual T-Pose from 3 different perspectives.

(a) Jessie model front

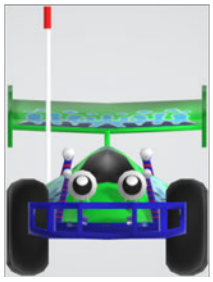


(b) Jessie model side

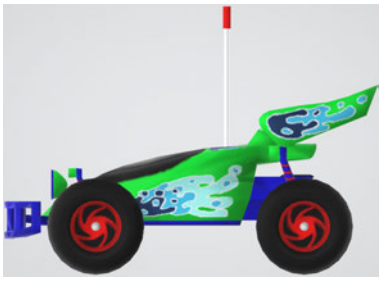
(c) Jessie model back

Figure 6.36: Jessie 3D model in T-Pose from 3 different perspectives.

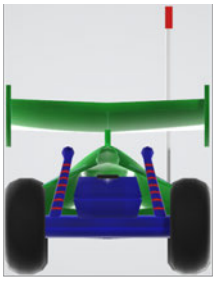
- RC: in this case we use a 3D model in .fbx format the. The model is available at the link. The figure below (figure 6.40) shows the model from 3 different perspectives.



(a) RC model front



(b) RC model side



(c) RC model back

Figure 6.40: RC 3D model from 3 different perspectives.

6.1.2 Interactive objects

There are some 3D models related to interactive objects. As introduced the interactive objects are the following:

- PSOne controller: is a model in .fbx format. It is possible to download the model at the link. The model represents the dualshock of Playstation 2. In order to ensure that the colours were consistent with those of Playstation 1, some modifications have been applied to this one in Unity. In the images (figure 6.43) below the model is represented in different perspectives.

(a) Dualshock model front

(b) Dualshock model side

Figure 6.43: Dualshock 3D model from 2 different perspectives.

- Door: is a model in .fbx format. You can download the model at the link. In the images (figure 6.46) below the model is represented in different perspectives.

(a) Door model front

(b) Door model side

Figure 6.46: Door 3D model from 2 different perspectives.

- Luxo Ball: is a model in .fbx format. You can download the model at the link. In the images (figure 6.49) below the model is represented in different perspectives.

(a) Luxo ball model front

(b) Luxo ball model side

Figure 6.49: Luxo ball 3D model from 2 different perspectives.

6.1.3 Andy's room

All the models and textures used to build Andy's room are available at the link. The models have been extrapolated and modified in Blender in order to be imported in Unity. All models are in .fbx format. The image below represents the room seen from different viewpoints.

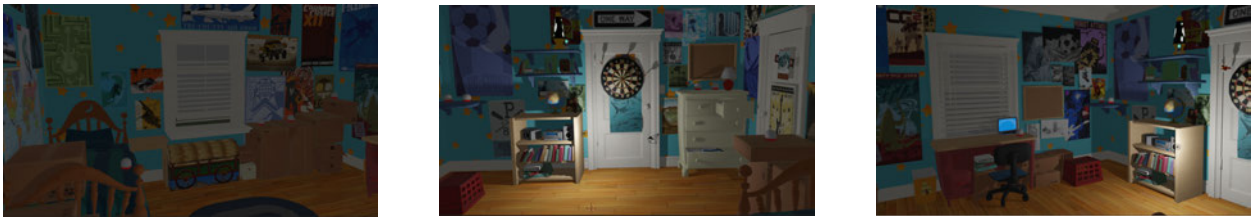


Figure 6.50: Room 3D model from 3 different perspectives.

6.2 2D sprites

The 2D Sprites have been created using some images found on the internet, some details have been added to them using the well-known IPad program called Procreate. All images are in PNG format.

Figure 6.51: Game's Logo

6.2.1 HUD

It is composed by: **Player UI** and **Dialog boxes**.

Player UI

Here are listed all the 2D sprites that the player can see on the screen while he is playing.

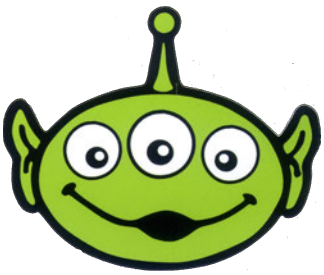
Collectible Icons Below, the list of all the collectible icons.

(a) Buzz (b) Jessie (c) Bullseye (d) Hamm (e) Rex (f) Slinky (g) RC

Figure 6.52: Icons of all the collectibles in the game

Dialog boxes All the 2D sprites concerning dialogs.

(a) Background (flipped for Woody's thoughts)



(b) Alien's icon used on the top of the tip dialog

(c) Woody's icon used on the top of the thought dialog

Figure 6.53: Sprites used for Dialog boxes with aliens and for Woody's thoughts

6.2.2 Menu UI

All the backgrounds used for menus are shown below.

Figure 6.54: Background used for both Main and Pause menus.

Figure 6.55: Background used for Game Over screen.

6.2.3 Skybox

For the skybox has been used a cubemap of a city.

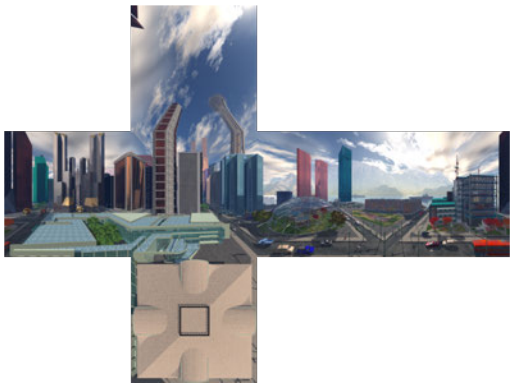


Figure 6.56: Skybox cubemap

6.3 Animations

To make objects animated we have used Mixamo that allows adding animations to the 3D models. In particular, the 3D models to which animations have been applied are those related to the Player and NPC.

6.3.1 Player

As introduced the player, the main character, is represented by Woody. Since Woody has to perform several actions, there are many animations that have to be applied to the model. In particular the animations can be divided according to Woody's skills including:

1. jumping;
2. climbing;
3. running;
4. walking.

There is also two additional animations, the first one that is applied in case Woody falls from a platform, or he releases his hands while attached to a ledge and the second one in case Woody is still standing.
Furthermore, in case of victory, an animation was used with the aim of making the model dance. The animation used is called "Gangnam Style" provided by Mixamo.

Jumping

Different animations are applied to the 3D model based on the type of jump. The jumping action performed by Woody can be of two types:

- stationary jump: when Woody jumps from stationary. In this case several animations are applied on the three phases of each jump, these are:

  1. jump: This is the first phase of the jump, when Woody jumps up from the ground. In this phase the Mixamo animation called "Jumping Up" has been chosen.
  2. floating: this is the second phase of the jump, when Woody once in the air begins to float. In this phase the Mixamo animation called "Floating" has been chosen;
  3. landing: this is the third phase of the jump, when Woody lands on the floor. In this phase the Mixamo animation called "Falling To Landing" has been chosen;;

- in motion jump: when Woody jumps while he's moving. In this case several animations are applied on the three phases of each jump, these are:

  1. jump: this is the first phase of the jump, when Woody jumps up from the ground. In this phase the Mixamo animation called "Jump" has been chosen.
  2. floating: this is the second phase of the jump, when Woody once in the air begins to float. In this phase the Mixamo animation called "Floating" has been chosen with a different posture compared to stationary jumping ;
  3. landing: this is the third phase of the jump, when Woody lands on the floor. In this phase the Mixamo animation called "Falling To Landing" has been chosen;

Climbing

Different animations are applied when Woody hangs on a ledge. Different animations are applied when Woody hangs on a ledge. The actions that Woody can do once he hangs on to a ledge are:

- hanging: when Woody is standing still while hanging on to a ledge. In this case the Mixamo animation called "Hanging Idle" has been chosen;

- hanging on movement: when Woody moves to the right or left while clinging to a ledge. In this case the Mixamo animation called "Left Shimmy" or "Right Shimmy" (depending from the direction) has been chosen;

- climbing: In this case we have two phase:

  1. climbing surface: when Woody climbs over a ledge. In this case the Mixamo animation called "Climbing" has been chosen;
  2. crouched walking: once Woody has climbed over the ledge he walks two steps forward on the surface. In this case, the Mixamo animation called "Crouched Walk" has been chosen;

Running

As introduced, one of Woody's skills is running, in this case the animation called "Drunk Run Forward" by Mixamo was chosen.

Walking

As introduced, one of Woody's skills is walking, in this case the animation called "Happy Walk" by Mixamo was chosen.

Falling

When Woody falls from a surface, two different animations are associated with the falling and the landing, which are respectively:

- the animation called "Falling Idle" by Mixamo;
- the animation called "Falling To Landing" by Mixamo;

Idle

A Mixamo animation called "Happy Idle" has been chosen in case Woody is standing still.

6.3.2 Mom

As introduced, Andy's mum performs a routine inside the room. Two phases have been defined for the routine:

- idle looking around: when Andy's mum stands still on the place and looks around. The animation called "Looking Around" by Mixamo;
- walking: When Andy's mum moves from one point to another in the room. The animation called "Walking" by Mixamo.

Furthermore, in case of victory, an animation was used with the aim of making the model dance. The animation used is called "Dancing Twerk" provided by Mixamo.

6.3.3 Alien

The animation of the alien depends on the proximity of woody. In particular, when:

- Woody enters the area of proximity of the alien, the animation called "Waving" by Mixamo has been chosen;
- Woody is outside the alien's proximity area, the animation called "Idle" by Mixamo has been chosen.

Furthermore, in case of victory, an animation was used with the aim of making the model dance. The animation used is called "Cheering" provided by Mixamo.

6.4 Visuals

6.4.1 VFX

The VFXs used in this game are mainly provided by the free unity package **Cartoon FX Free**. Some effect has been customized to better fit our porpose.

Dust Run This effect is used to simulate the dust raised by player when he's running. The prefab used for this effect is *CFX3_Hit_SmokePuff* provided by **Cartoon FX Free** package.

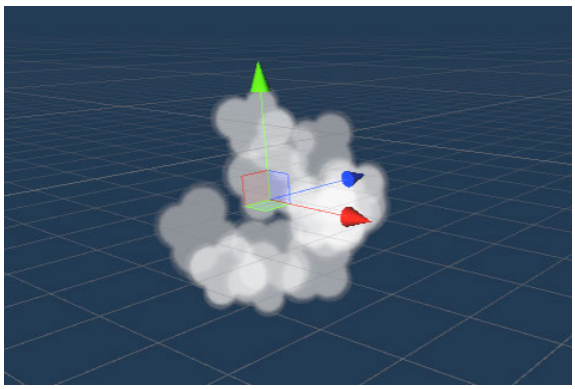


Figure 6.57: Picture of the *Dust Run* effect

Dust Landed This effect is used to simulate the dust raised by player when he's landing on the ground from a jump or a fall.
The original prefab used for this effect is *CFX3_Hit_SmokePuff* provided by **Cartoon FX Free** package.

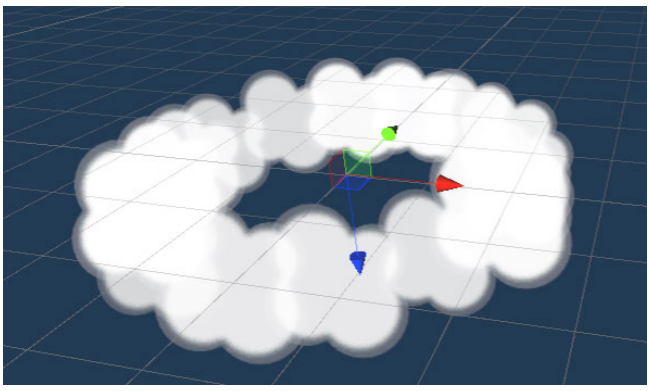


Figure 6.58: Picture of the *Dust Landed* effect

Item Collected Puff This effect is played when an item is collected and disappears. The original prefab used for this effect is *CFX_MagicPoof* provided by **Cartoon FX Free**

package.

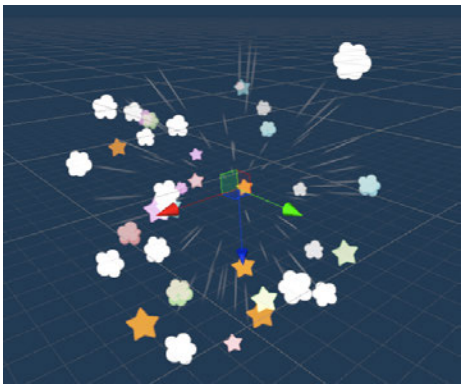
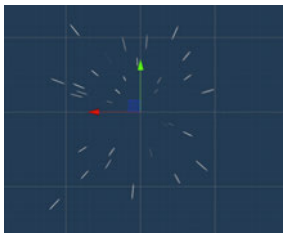
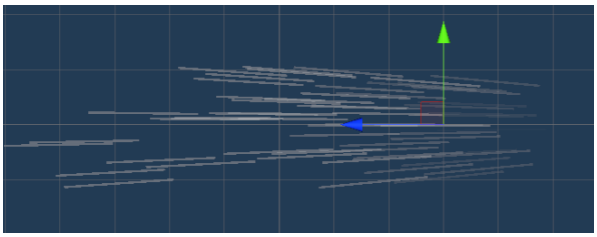


Figure 6.59: Picture of the *Item Collected Puff* effect

High Speed This effect has been created from skratch using Unity's Particle Systems. Here is a link to the video used as guide to create this effect: Link.



(a) Front view

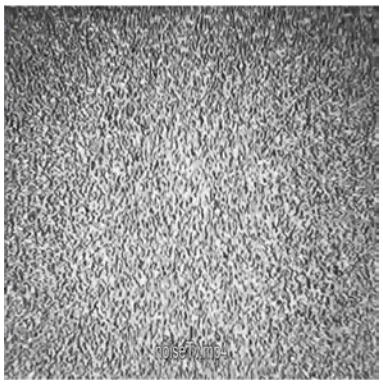


(b) Side view

Figure 6.60: Pictures of the *High Speed* effect

6.4.2 Video clips

In this game there are only 2 videoclip, both displayed on the TV screen.
The first is "noiseTV.mp4", a 2 seconds clip used to display tv screen noise.
The second is "PSone_startup.mp4", clip that starts when the PSOne is switched on. This one lasts 35 seconds and contains 2 merged clips. At the beginning ther is the PSOne startup intro (clip downloaded from the internet) and then a custom clip used to give information to the player about the imminent arrival of Andy's mom's.



(a) “noiseTV.mp4”

(b) “PSone_startup.mp4”

Figure 6.61: Pictures representing the videoclips preview

6.5 Audio

To ensure that the mood of the game reflects Disney standards, we made a very careful selection of audio clips. In particular, audio clips from other sites or recorded directly by us were used.
For all audio clips we did post processing using the well-known Audacity tool, which allowed us to improve the audio in terms of intonation, volume and balance between bass and treble. We made a distinction between two main types of audio clips:

- music: used for game menus, background and final scene music in the game;
- sounds effects (SFX): used to bring different characters or objects within the game to life on a sound level.

6.5.1 Musics

As introduced by music we mean all audio clips used for background music related to:

- menu: the audio clip in this case is an instrumental on the piano of the famous main theme of the Toy Story saga titled ”You’ve got a friend in me” performed by Randy Newman . You can listen the entire audio clip at the link;
- ingame: in this case the audio clip is the Italian version of the main theme (”You’ve got a friend in me” by Randy Newman) called ”Hai un amico in me” sung by Riccardo Cocciante. The choice of the Italian variant is not accidental but is dictated by the fact that it represents a great piece of our childhood. You can listen the entire audio clip at the link;
- final scene: in this case, since the final scene, the victory scene, presents the dancing animated characters, a remix of the main theme of the toy story (”You’ve got a friend in me” by Randy Newman) was chosen as audio clip. The complete audio clip can be heard in full at the link.

6.5.2 SFX

The sound effects consist of all the audio clips used to handle:

- interaction of the player with the collectables;
- player actions;
- player interaction with door and PSOne;
- interaction of the player with aliens;

Player interaction with collectables

As introduced, inside Andy's room there are some collectables, for each of these there are two types of audio clips:

- audio clip concerning the collection. The audio clip name used within the project is "smokePoof.waw";
- audio concerning the interaction with the collectables: in this case a distinction must be made between the different collectables because, for each of them, a typical phrase has been recorded and a modification has been made on the tone and on the equalization of bass and treble through Audacity:
  - **Buzz Lightyear**: the voice actor is Davide Amato, the phrase that is reproduced after the interaction is "Sono Buzz Lightyear, space ranger!". The audio clip name used within the project is "buzz_lightyear.waw";
  - **Jessie**: the voice actor is Elisa Gallo, the phrase that is reproduced after the interaction is "hiyaa!". The audio clip name used within the project is "jessie.waw";
  - **Hamm**: the voice actor is Davide Amato, the phrase that is reproduced after the interaction is "hey, sono Hamm!". The audio clip name used within the project is "hamm.waw";
  - **Rex**: the voice actor is Davide Amato, the phrase that is reproduced after the interaction is "Eccomi, sono Rex!". The audio clip name used within the project is "rex.waw";
  - **Slinky Dog**: the voice actor is Armando Pezzimenti, the phrase that is reproduced after the interaction is "wof, wof, wof!". The audio clip name used within the project is "slinkydog.waw";
  - **Bullseye**: in this case an audio clip playing a galloping horse was used. The audio clip name used within the project is "bullseye.waw";
  - **RC**: in this case an audio clip playing the sound of an accelerating radio-controlled machine was used. The audio clip name used within the project is "rc.waw";

Player actions

Different audio clips were used for Woody according to his skills:

- **walking** or **running**: in this case an audio clip found on freesound was used. The audio clip name used within the project is "step.waw";
- jumping: when Woody jumps from stationary or in motion. In this case there are three phases:
  - **jump**: this is the first phase of the jump, when Woody jumps up from the ground. In this case an audio has been recorded and modified via Audacity to have better pitch and equalization of treble and bass. The voice actor for this audio clip is Armando Pezzimenti. The audio clip name used within the project is "hop.waw";
  - **floating**: this is the second phase of the jump, when Woody once in the air begins to float. In this case an audio found on Storyblocks has been used. The audio clip name used within the project is "wind.waw";
  - **landing**: this is the third phase of the jump, when Woody lands on the floor. In this case an audio found on Storyblocks has been used and it has been modified through Audacity to have a better pitch and equalization of treble and bass. The audio clip name used within the project is "Landed.waw";
- **climbing**: ifferent audio clipls are applied when Woody hangs on a ledge. The actions that Woody can do once he hangs on to a ledge are:
  - **hanging on movement**: when Woody moves to the right or left while hanging on a ledge. In this case an audio clip found on freesound was used. The audio clip name used within the project is "Hang.waw";
  - **climbing**: when Woody climbs over a ledge. In this case an audio has been recorded and modified via Audacity to have better pitch and equalization of treble and bass. The voice actor for this audio clip is Armando Pezzimenti. The audio clip name used within the project is "oplà.waw";
- **falling**: when Woody falls from a surface, two different audio clips are associated with the falling and the landing, which are respectively:
  - in this case an audio found on Storyblocks has been used. The audio clip name used within the project is "wind.waw";
  - in this case an audio found on Storyblocks has been used and it has been modified through Audacity to have a better pitch and equalization of treble and bass. The audio clip name used within the project is "Landed.waw";
- **ragdoll**: when Woody hits something while he is in ragdoll mode, an audio clip is played. In this case an audio found on Storyblocks has been used and it has been modified through Audacity to have a better pitch and equalization of treble and bass. The audio clip name used within the project is "Landed.waw";

Player interaction with door and PSOne;

In this case we have audio clips specific to Woody's interaction with the door or PSOne:

- when Woody interacts with PSOne before collecting all the collectables an audio clip is played. In this case an audio has been recorded and modified via Audacity to have better pitch and equalization of treble and bass. The voice actor for this audio clip is Armando Pezzimenti. The audio clip name used within the project is "ps_thought.waw";
- when Woody interacts with the door before Andy's mum has entered the room an audio clip is played. In this case an audio has been recorded and modified via Audacity to have better pitch and equalization of treble and bass. The voice actor for this audio clip is Armando Pezzimenti. The audio clip name used within the project is "dialog_door.waw";

Player interaction with aliens

In this case you have different types of audio clips, which are played when:

- Woody is in proximity of an alien: in this case an audio has been recorded and modified via Audacity to have better pitch and equalization of treble and bass. The voice actor for this audio clip is Elisa Gallo. The audio clip name used within the project is "alienHello.waw";
- Woody interacts with the alien to hear what he wants to say. In this case there are two phases:
  - beginning of the conversation: in this case an audio has been recorded and modified via Audacity to have better pitch and equalization of treble and bass. The voice actor for this audio clip is Elisa Gallo. The audio clip name used within the project is "alienWooo.waw";
  - transition to the alien's next sentence: in this case an audio found on Storyblocks has been used. The audio clip name used within the project is "alienSqueeze.waw";

Implementation

In the following sections we'll go into the details about the implementation of the game. We'll do a deep view of all those aspects discussed so far.
Before to start coding, we settled down the project by installing the same Unity version (2019.3.7f1) and we overwrote 2 unity template scripts so we both work with the same code structure and without the need to add every time the same regions inside the code. The overwritten templates in this project are "*81-C# Script-NewBehaviourScript.cs.txt*" and "*83-C# Scriptable Object-NewScriptableObjectScript.cs.txt*". Here is a guide on How to customize Unity script templates.

7.1 External packages

To realize this project, many external packages have been installed and used. Here is a list with all the additional packages installed, with a short description for each of them.

7.1.1 Cinemachine

This package nowadays is a standard in game development with unity. It offers state-of-the-art solutions to implement a wide range of different and complex camera behaviours and mechanics. This package is so advanced and complete that it's used not only for games, but also for high quality short films. It has been used to implement the main **Third Person Camera** and many other camera aspects (like **camera's animations**).

More info about this package can be found at link.

7.1.2 Unity's New Input System

In this project, Inputs are mainly managed through the Unity's **New Input System**. This system is able to create an abstract layer where we define "***Actions***" (i.e. Shoot, Move, Jump, etc...) and for each *Action* we can add "***Bindings***", that are the real triggers (i.e. a keyboard button pressed/released/holded, a gamepad stick movement, a gamepad trigger pressed, etc...) that "*activate*" that *action*. So it is easy to imagine that we can define the actions once and implement it as many time and with as many devices as we want.

A set of input actions regarding the same domain (for instance, all the actions regarding the player's movement) can be *encaspulated* in the same "***Action Map***". For each *Action Map*, an interface is automatically generated. In this way, a script could implement the interface of a particular action map to implements the **callback functions** relative to its actions.

More info about this package can be found at link

7.1.3 Unity Core Library

This package offers different core components and scripts to facilitate game development by offering *easy-to-use* solutions to implement common features. Some of the features included in this package are: *Event System, Timer, Object Pooling, Undo command,* etc...
During the development of the project, an extensive use of the *Event System* has been made, so it deserve further investigation.

Event System

This system implements a sort of *Observer Pattern*. It is very simple to use and can be used with different approaches. The approach used in this project involves several actors:

- **GameEvent**: using the *Observer* termonology, it is the **Subject**. It is responsible to communicate to *observers* whenever the event is *raised*.
- **Observer**: method that implements the logic that must be executed whenever the event it is *listening to* is *raised*.
  Note: there could be many *Observers* for the same **GameEvent**
- **Raiser**: method that raises the event.
  Note: there could be many *Raisers* for the same **GameEvent**.

GameEvent

A GameEvent is a *Scriptable Object* that can be created by going to *Asset>Create>Core*[1]*>Event System>GameEvent*. The default GameEvent doesn't allow to pass parameters on the event raising. In order to pass a parameter on event raising, **Unity Core Library** offers a set of parameterized GameEvents among which we can find **GameEventInt, GameEventString, GameEventFloat, GameEventBool** and others.
Moreover, if you need to pass a **custom object** as parameter, you can create a class *GameEventCustomObject* that extends *GameEvent<CustomObject >*.

Observer

The Observer, as mentioned above, is a method. To better explain how it works, let's do an example. Let's say to have a GameEvent called **deathEvent**. We want to call the

[1] **Core** section under *Asset>Create* is not visible by default. It is visible after the installation of **Unity Core Library** package

method **OnDeath()**, that is inside the script **PlayerManager** (that extends *MonoBehaviour*), whenever **deathEvent** is *raised*. We have to *subscribe* the observer **OnDeath** to **deathEvent**.
To do so, we can call **deathEvent**.*Subscribe*(**OnDeath**) inside **PlayerManager**.*OnEnable*(), and we can also call **deathEvent**.*Unsubscribe*(**OnDeath**) inside **PlayerManager**.*OnDisable*() in order to unsubscribe the observer when we don't need it anymore.

Raiser

Whenever we want to raise an *event*, we can just call ***eventName.Raise()*** in whatever method of whatever script we want.

More info about this package can be found at link

7.1.4 Cartoon FX Free

Cartoon FX Free is a package that can be downloaded for free from the Unity Asset Store. It offers a lot of high-quality **vfxs** for many scenarios.

More info about this package can be found at link

7.1.5 TextMesh PRO

This package offers advanced solutions to implement **TextMeshes** and **UI Texts**, offering the possibility to fully customize them. It has been used to implement the game UI.

More info about this package can be found at link

7.1.6 Pro builder

This package is mainly used for prototyping. It offers advanced tools to create and edit custom geometry in unity in an easy way. It has been widely used in the first phase of this project, to implement the core features of the game.

More info about this package can be found at link

7.2 Player

As introduced Woody is the main character used by the player. In particular, we have made a division between:

- **Prefabs**: the prefab related to Woody;

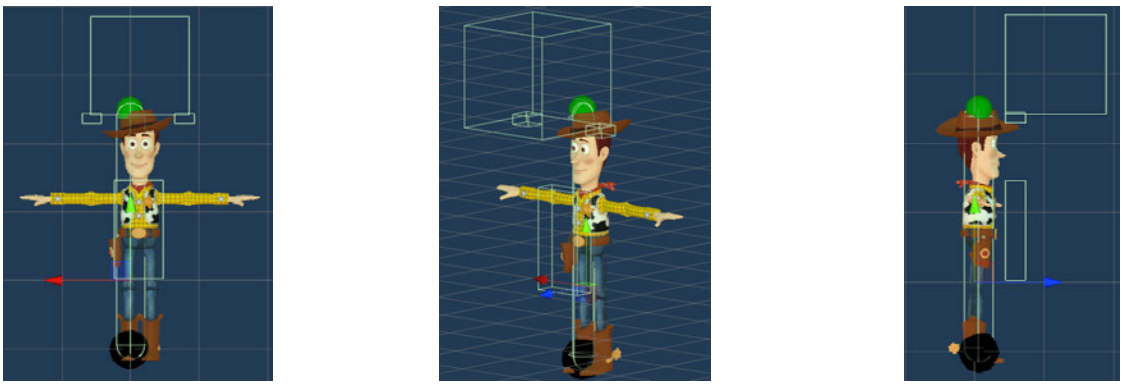
- **Scripts**: all scripts associated to Woody to attribute behaviours to it;
- **Animator**: the animator used to associate the different animations introduced to Woody;
- **InputAction maps**: the maps used to bind player inputs.

7.2.1 Prefabs

Woody, the main character, has several prefabs associated, which are:

- WoodyAlive;
- WoodyRagdoll;
- WoodyDancer;

WoodyAlive



(a) front (b) 45° (c) side

Figure 7.4: WoodyAlive's isometric visualization. We can notice the CharacterController's CapsuleCollider, various BoxColliders used for climb checks, a green sphere (SphereCast for ceiling check) and a black one (SphereCast for ground check).

WOODYAlive is the prefab that is controlled by the **CharacterController** (during the level) that allows the Player to perform actions with Woody when it is not in "*ragdoll*" mode. The following figure shows the internal structure of the prefab.

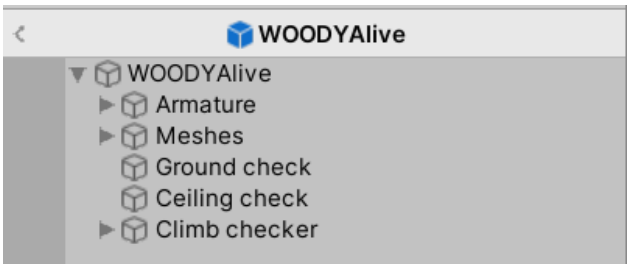


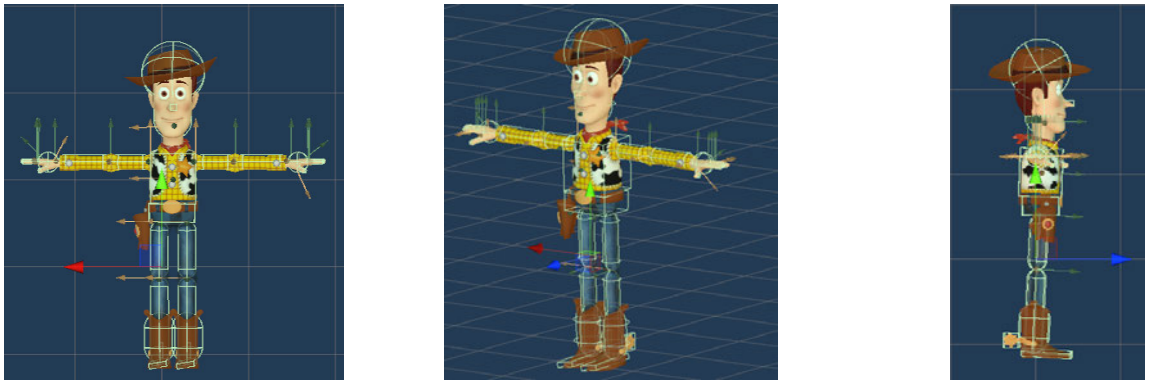
Figure 7.5: WoodyAlive's structure

As we can see in the figure, the root of this hierarchy is represented by "WOODYAlive". The components inside this prefab are:

- **Character Controller**
- **Animator**
- **Player Controller (Script)**
- **Animator Controller (Script)**

Since, as introduced Woody's model is rigged, the object contains an **Armature** representing the skeleton (without colliders) and **Meshes** that contain all the materials related to the model. Then, there are **Ground**, **Ceiling** and **Climb checker** that are used for the related checks.

WoodyRagdoll



(a) front (b) 45° (c) side

Figure 7.9: WoodyRagdoll's isometric visualization. We can notice all the colliders used to implement the *ragdoll effect*.

This is the prefab used when Woody is in "*ragdoll*" mode. As shown in the picture below, it has a structure simpler than the *woodyAlive*'s one but it has an **Armature** full of *primitive*

colliders (almost one per bone), each of which has been accurately customized in order to implement in a effective and efficient way the "ragdoll effect".

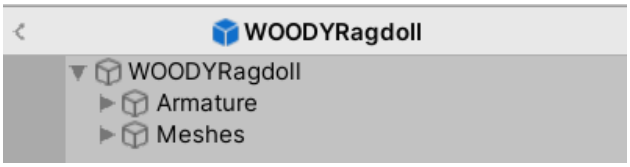


Figure 7.10: WoodyRagdoll's structure

WoodyDancer

This model has exactly the same structure as **WoodyRagdoll** but with some small difference. The entire armature is collider free and the root has an **Animator** component attached that makes it dance.

7.2.2 Scripts

In order to manage all actions related to Woody, we have defined several scripts. In particular we have divided the scripts into 6 macro areas:

- **Movement** related scripts;
- Scripts related to **selection** and **interactions**;
- Scripts related to the management of the **animator**;
- **Graphic** and **sound effects** scripts;
- **Mode change** scripts (**Alive/Ragdoll/Hideout**);
- **Utility** scripts;

Movement related scripts

As can be seen from the figure (Figure 7.11), the two scripts used to implement movement (*PlayerController.cs* and *HideoutController.cs*) represent two **concrete** classes that:

- *extend* the **AbstractCharacterController** abstract class that, in turn, implements the **ICharacterController** interface;
- *implement* the **InputActions.IFootControlsActions** interface;

Moreover, **PlayerController.cs**, implements also **InputActions.IClimbingControlsActions** interface.

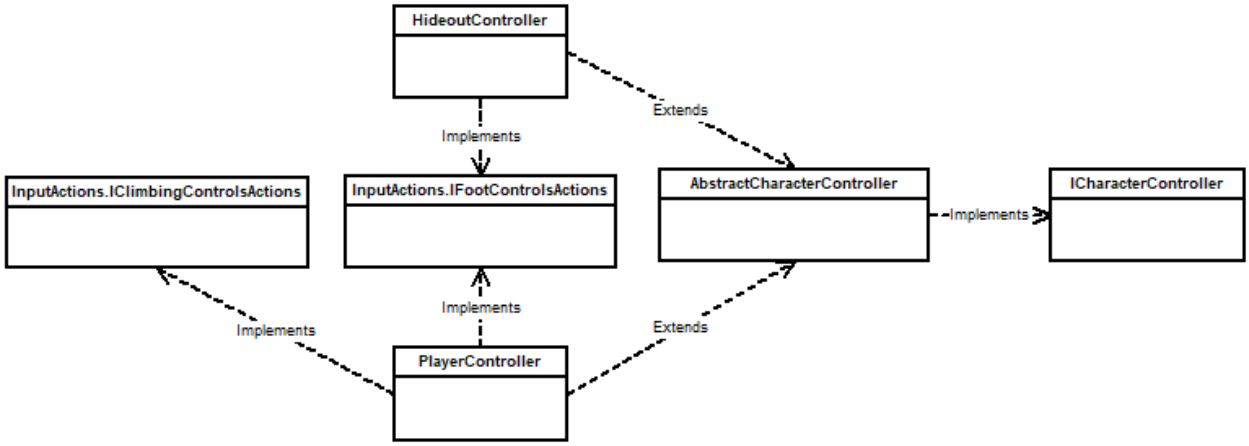


Figure 7.11: Class diagramm of the "movement related" Controllers scripts

A distinction can be made between scripts used for movement based on Woody's mode:

- **PlayerController.cs**: this class defines how the player should move when it is in "*alive*" mode. This script must be attached directly to the player object that has to be moved. Events for which this script has a callback method:
  - *tipEnabled* and *tipDisabled*: callbacks that disable/enable respectively player movements.
  - *disableMovementForSeconds*: callback that disable movements only for some seconds. This is done in order to disable movements when woody interact with the playstation.
- **HideoutController.cs**: This class defines how the player should move when it is in "*hidden*" mode. It must be attached directly to the object within which the player can hide;

ICharacterController.cs

This script contains:

- **ICharacterController**: **interface** that defines some basic methods that must be implemented to move a character.
- **AbstractCharacterController**: **abstract** class that *extends* **MonoBehaviour** and *implements* **ICharacterController**. This abstract class defines the basic movement behaviours of a character. Moreover, it assumes that the character has a **CharacterController** component attached (internally used by this class to move the character).

ClimbCollisionDetection.cs

This script is used to perform various checks in order to establish if the player can do actions regarding climbing movements.

LateralHanghingController.cs

This script is used to communicate to other scripts when, during hanging movement, a single lateral step is executed. To do so, internally it uses a timer. Events raised by this script:

- *LateralStepExecuted*: event raised when the timer is expired and a single lateral step, in hanging mode, has been executed.

Scripts related to selection and interactions

Regarding the selection of objects, a class called **SelectionManager** has been defined together with a hierarchy in which there are two *concrete* classes, **OutlineSelectionResponse** and **HighlightSelectionResponse** that implement **ISelectionResponse** interface (Figure 7.12).

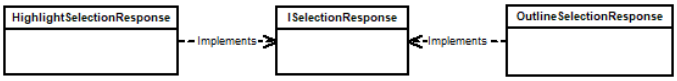


Figure 7.12: Hierarchy related to the classes defined in the scripts that manage the player mode change.

In order to complete this task, a video published on YouTube by Infallible Code was used as a reference, which consists of two parts (Part 1, Part 2).
Moreover a script called *InteractionManager.cs* (described below) has been defined for the management of interactions and the collection of other objects by the player.

ISelectionResponse.cs

This script defines an interface that once implemented requires the definition of two useful methods for selecting and deselecting objects.

OutlineSelectionResponse.cs

This class is used to highlight the outlines of selected objects or to finish highlighting objects that are no longer selected. In order to accomplish this task, two methods called *OnDeselect* and *OnSelect* are defined within the class.

SelectionManager.cs

In order to accomplish this task, this class uses the methods of the class described above (OutlineSelectionResponse). It uses a layermask in order to select layers with which the player can interact.

InteractionMager.cs

This script defines the concrete **InteractionMager** class. Using the two scripts introduced, it is possible to manage the interaction and the collection of objects.
Events raised by this script:

- *interactWithCollectible*: raised when the player *interact* with a *collectible* object.

- *interactWithAlien* and *alienInteraction*: raised when the player *interacts* with an alien.
- *ThoughtTriggered*:raised when the player *interacts* with some object that is locked at the moment.
- *onCollect*: raised when the player *collects* a *collectible* object.
- *PSOneOpened* and *disableMovementForSeconds*: raised when the player *interacts* with the joypad and TV is switched on.

Scripts related to the management of the animator

To manage the animator, a script called *AnimatorController.cs* has been defined. This script defines the concrete **AnimatorController** class.

AnimatorController.cs

The defined class is used to manage the different animations transactions to be communicated to the Animator. To perform this task this class uses global information shared by the player via *PlayerSharedInfo.cs* and *GameSharedInfo.cs*.
Using the two scripts introduced, it is possible to perform controls to ensure the Animator (WoodyALIVE component) a consistent transition to the next animation.
The managed actions allow to raise the following events:

- *playerJumped*: raised when the "jump" animation is launched;
- *playerLanded*: raised when the "land" animation is launched;
- *surfaceClimbed*: raised when the "climb" animation ends;
- *hanged*: raised when the "hang" animation is launched;

Graphic and sound effects scripts

SFXPlayerController.cs

To manage the sounds coming from the player, the SFXPlayerController.cs script has been defined. The aim of this script is to correctly manage the AudioClip reproduction through the use of three AudioSource:

- _audioSourceMotion, is used for:
  - the jump;
  - landing;
  - walking;
  - all phases related to climbing and hanging;
  - ragdoll (when Woody touches the ground);

- _audioSourceSpeach, is used for the reproduction of Woody's thoughts:
  - one related to Woody's thought once he interacts with the playstation when not all the collectibles have been collected yet, related to the AudioClip called "ps_thought.waw";
  - the one related to Woody's thought once he interacts with the door when it is still closed, related to the AudioClip called "dialog_door.waw";

Events for which this script has a callback method:

- *stepWalkExecuted*: callback that plays the "walk" audioclip.
- *stepRunExecuted*: callback that plays the "run" audioclip.
- *playerJumped*: callback that plays the "jump" audioclip.
- *playerLanded*: callback that plays the "landed" audioclip.
- *startCamShake*: callback that plays the "wind" audioclip.
- *stopCamShake*: callback that stops the "wind" audioclip.
- *hanged*: callback that plays the "hang" audioclip.
- *lateralStepExecuted*: callback that plays the "hang" audioclip.
- *surfaceClimbed*: callback that plays the "climbSurface" audioclip.
- *ragdollCollisionDetected*: callback that plays the "ragdollHit" audioclip.
- *thoughtTriggered*: callback that plays the right "thought" audioclip based on the parameter received.

VFXPlayerController.cs

This script manage the activation/deactivation of **visual effects** related to the player. Events for which this script has a callback method:

- *stepRunExecuted*: callback that plays the "dustRunning" particle system.
- *playerLanded*: callback that plays the "dustLanding" particle system.

Events raised by this script:

- *startCamShake* and *startHighSpeedFX*: raised when the ragdoll velocity reaches a high value.
- *stopCamShake* and *stopHighSpeedFX*: raised when the ragdoll velocity reaches a low value.

Mode change scripts (Alive/Ragdoll/Hideout)

Scripts have been defined to manage the player's mode changes. In particular, in the figure (Figure 7.13) below is defined the hierarchy used for the scripts related to the actions introduced. Two concrete classes called **RagdollModeChanger** and **HidingModeChanger** which extend the **AbstractModeChanger** abstract class have been defined.
In addition:

- the **RagdollModeChanger** implements the **InputActions.IRagdollControlsActions** interface;
- the **HidingModeChanger** implements the **InputActions.IHideoutControlsActions** interface;

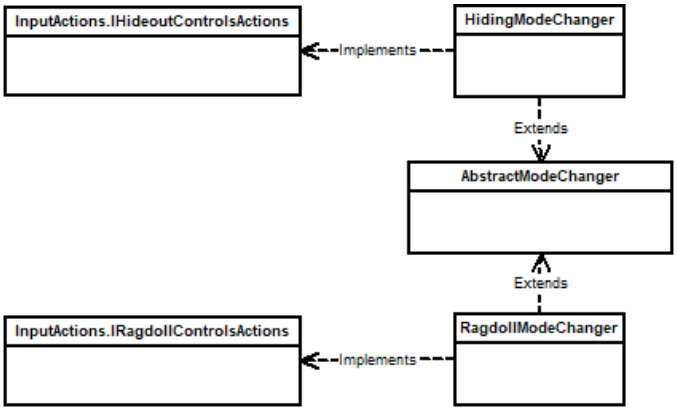


Figure 7.13: Hierarchy related to the classes defined in the scripts that manage the player mode change.

AbstractModeChanger.cs

It is an abstract class used to switch from Woody's alive mode to a second mode. This class takes into account the Player in **PlayerSharedInfo** and the Camera information.
It defines a method that allows to instantiate the player's prefab in alive mode. It also defines another method that is overridden by the concrete classes that extend this class to ensure the correct transition to secondary mode.

RagdollModeChanger.cs

It is a concrete class that defines the player mode change from Ragdoll to Alive and vice versa. It also allows:

- to define Woody's speed once it goes into Ragdoll mode by applying a force to it;
- to instantiate Woody in the correct position once the switch from Ragdoll to Alive mode is performed.

Events for which this script has a callback method:

- *tipEnabled* and *tipDisabled*: callbacks that disable/enable respectively the Ragdoll input actions;

HidingModeChanger.cs

It is a concrete class that defines the player mode change from Hideout to Alive and vice versa. In addition:

- it allows to activate the HideOutController introduced in order to allow the player the actions related to the current game mode;
- it performs operations on the camera in order to adapt the camera to the current game mode;
- it allows to instantiate Woody in the correct position once the switch from Ragdoll to Alivemode is performed.

Events raised by this script when player mode switches to Hidden:

- *StopCamShake* and *StopHighSpeedFx*;

Utility scripts

Some scripts related to the sharing of some main information about the player have been defined.

CharacterControllerConfiguration.cs

It defines a class containing character's **base default settings** such as *gravity*, *movement and rotation speed*, *movement, ground and ceiling checks constraints* and so on. It allows to customize these values for every character that has a concrete extension of **AbstractCharacterController** attached.

PlayerSharedInfo.cs

The class defined in this script is a *Scriptable Object* and contains information about **player's current state**. An instance of this class works like a **global reference**. It is used to share, among different classes, information about player's current state.
In addition, a script has been defined for Woody's collisions with the ground once the mode has been changed from Alive to Ragdoll.

HandleRagdollCollision.cs

This script is used to detect whenever the player collides with something in "*ragdoll*" mode. This script must be attached on every bone on which the collision must be detected. This script has been attached only to the main bones. Events raised by this script:

- *ragdollCollisionDetected*: raised when the ragdoll hits something with a sufficient velocity and if the last hit happaned not too close in time.

StepsEventController.cs

This script is used to communicate to other scripts when a single walk/run step is executed. To do so, internally it uses 2 timer, one for each kind of step. Events raised by this script:

- *stepWalkExecuted* and *stepRunExecuted*: raised when the timer for the relative kind of step expires.

7.2.3 Animator

For the management of transitions between Woody's different animations an Animator has been defined. As introduced, the **AnimatorController.cs** script is used to notify the Animator of the state transition of Woody's animations.
As can be seen from the figure (Figure 7.14), the animator is defined by a basic state machine which is responsible for managing the correct transaction between some animations (idle, walk and jump) and for delegating the transitions between the remaining animations to sub-states machines.

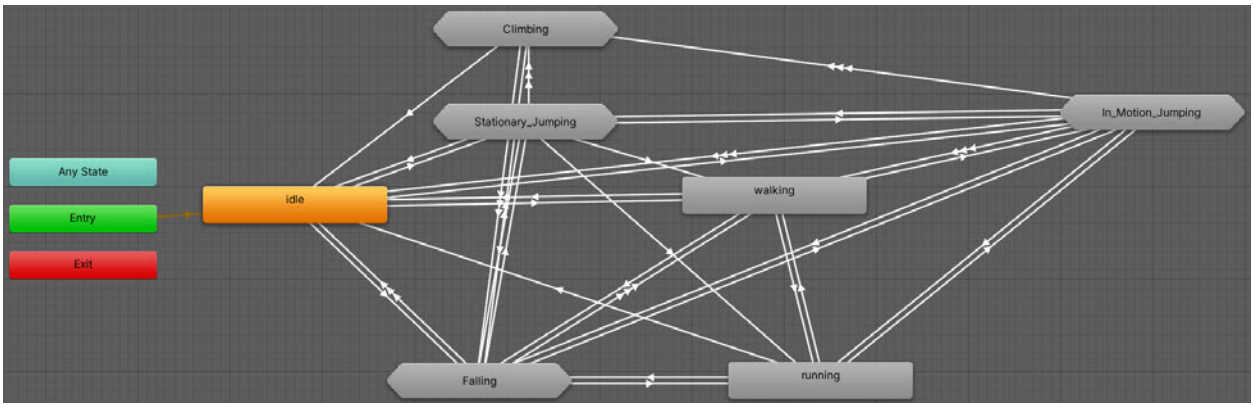


Figure 7.14: Basic state machine used for the animator

Each sub-state machine used has been defined for the management of the animation transitions related to the following actions:

- Stationary jumping, in order to manage this action the relative sub-machine defines what transitions must be made between the animations before delegating the task to the basic state machine. In particular the animations related to this action are:

  1. stationary jump;
  2. stationary floating;
  3. landing;

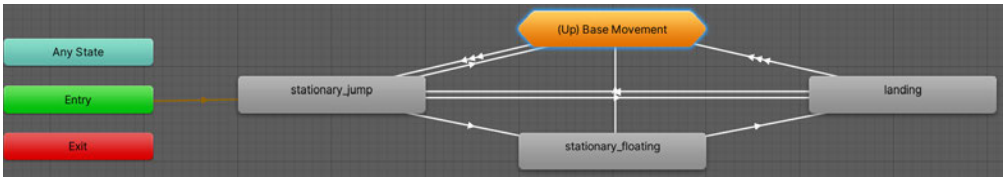


Figure 7.15: Sub-state machine used to manage animations related to stationary jumping

- In motion jumping, in order to manage this action the relative sub-machine defines what transitions must be made between the animations before delegating the task to the basic state machine. In particular the animations related to this action are:

  1. in motion jump;
  2. in motion floating;
  3. landing;

  ;

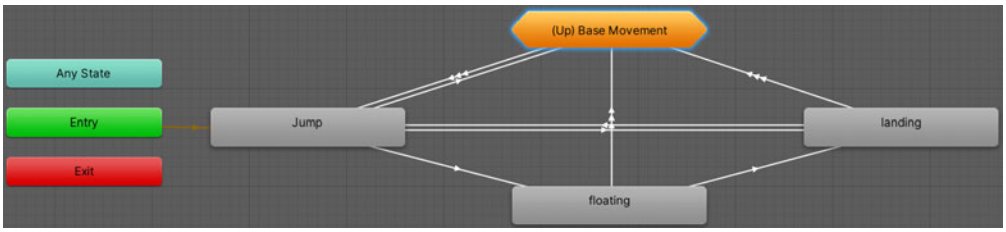


Figure 7.16: Sub-state machine used to manage animations related to in motion jumping

- Falling, in order to manage this action the relative sub-machine defines what transitions must be made between the animations before delegating the task to the basic state machine. In particular the animations related to this action are:

  1. falling;
  2. landing;

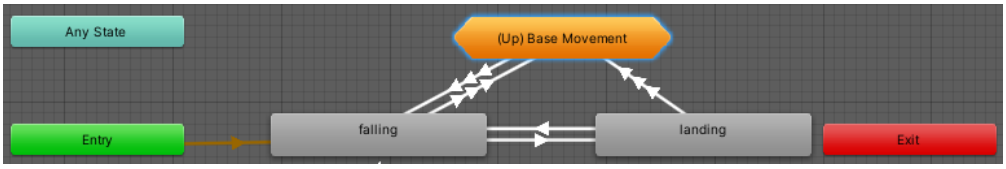


Figure 7.17: Sub-state machine used to manage animations related to falling

- Climbingin order to manage this action the relative sub-machine defines what transitions must be made between the animations before delegating the task to the basic state machine. In particular the animations related to this action are:

1. Hanging idle;
2. Lateral hanging (right or left);
3. Climbing;
4. Crouced Walk;

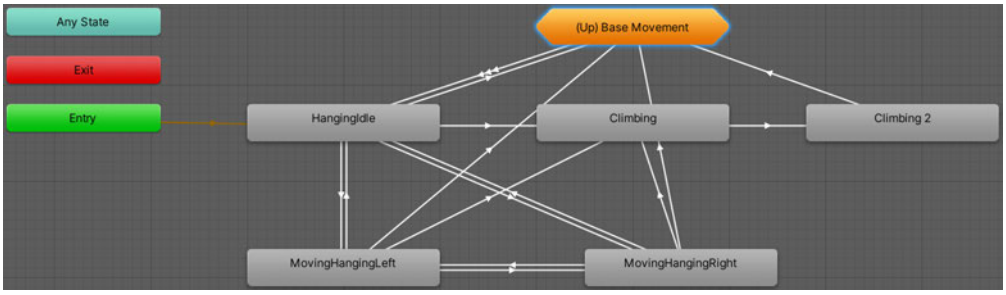


Figure 7.18: Basic state machine used to manage animations related to the animator

7.2.4 InputAction maps

As first step we created an **Input Actions** asset (Toolbar/Assets/Create/Input Actions). Then we added the following *Action Maps* and relative *Actions* (see section 7.1.2 for more info about Unity's new Input System).

Action Map: Foot controls

Action map that contains all the actions relative to player movements when the player is on foot. In code it's referenced as **InputActions.IFootControlsActions**:

- **Move**: action raised when player gives inputs to move on the x,z axis.
- **Jump**: action raised when player gives input to jump.
- **Run**: action raised when player gives input to run.

Action Map: Look controls

Action map that contains all the actions relative to player view. In code it's referenced as **InputActions.ILookControlsActions**:

- **Look**: action raised when player gives input to change view.
- **Zoom**: action raised when player gives input to zoom in/out his view.

Action Map: Ragdoll controls

Action map that contains all the actions relative to the player's ragdoll mechanic. In code it's referenced as **InputActions.IRagdollControlsActions**:

- **Ragdoll**: action raised when player gives input to switch between "*alive*" and "*ragdoll*" modes.

Action Map: Interaction controls

Action map that contains all the actions relative to the interaction of the player with "*interactable objects*" (3.3.3). In code it's referenced as **InputActions.IInteractionControlsActions**:

- **Interact**: action raised when player gives input to do a simple interaction.
- **Collect**: action raised when player gives input to *collect* a *collectible object*.

Action Map: Hideout controls

Action map that contains all the actions relative to the "*hidden*" mode. In code it's referenced as **InputActions.IHideoutControlsActions**:

- **Hide**: action raised when player gives input to switch between "*hidden*" and "*alive*" modes.

Action Map: Climbing controls

Action map that contains all the actions relative to climbing movements. In code it's referenced as **InputActions.IClimbingControlsActions**:

- **Release**: action raised when player gives input to leave the grip from hanging position.

Action Map: UI

Action map that contains all the actions relative to inputs for UI components. This action map was already available when the InputAction asset was created. In code it's referenced as **InputActions.IUIActions**:

- **Submit**: action raised when player gives input to submit/confirm. This is the only action used in this demo.

7.3 Jennifer (Andy's Mother)

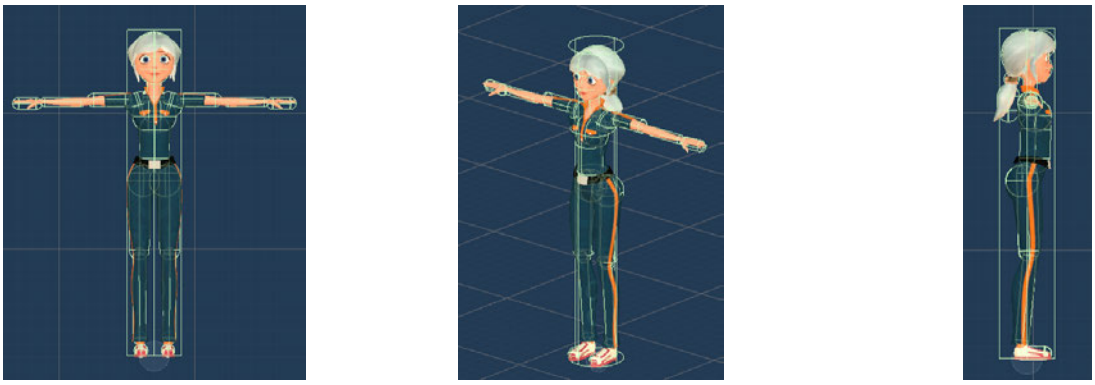
The structure of this section is the following:

- **Prefabs**: the prefab related to Andy's mother;
- **AI**: explanation of the artificial intelligence implemented for Jennifer.
- **Scripts**: all scripts associated to Jennefer to attribute behaviours to it;
- **Animator**: the animator used to associate the different animations introduced for Jennifer;

7.3.1 Prefabs

Two prefab has been made for Jennifer: **MOM** and **MOMDancer**.

MOM



(a) front (b) 45° (c) side

Figure 7.22: MOM's isometric visualization. We can notice the colliders and the cilinder representing the NavMesh Agent component

MOM's prefab has an unchanged structure compared to the original imported model. It contains an **Armature** representing the skeleton (with colliders added manually) and **Meshes** that contain all the materials related to the model. The following components have been attached to the root of this prefab:

- **Animator**
- **NavMesh Agent**
- **Mom AI (Script)**
- **FOV (Script)**

MOMDancer

This prefab is the same as the original model with only Armature (without colliders) and Meshes. The only addition is an Animator to make her dance in the end credits.

7.3.2 AI

The **artificial intelligence** implemented for Jennifer is very simple. At the beginning she stays standing outside the door, waiting to hear some loud sound. When the player open the PS1, after some seconds (left to the player to let him hide himself) she enters in the room and starts a **routine**. Having a set of locations to check, called "*waiting positions*" (Figure 7.23), this routine is structured in the following way:

1. she **randomly choose** one location to check among the "*waiting positions*".
2. she **goes** to the **chosen location**.
3. Once there, she stays in the reached location for a **random time**.

Moreover, she can detect the player (if he is in *Alive* mode) by using her **Field Of View**.

NavMesh, NavMesh Agent and NavMesh Obstacles

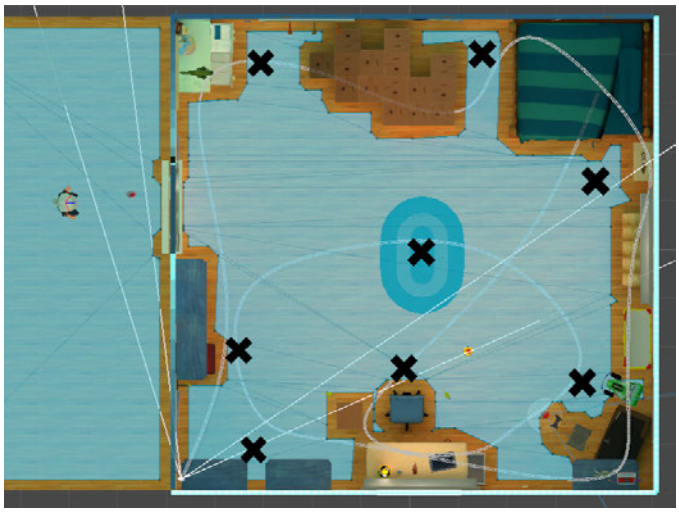


Figure 7.23: Isometric top-down view of the NavMesh. We can notice, marked with a black 'X', all the locations used by Jennifer's routine.

To implement Jennifer movements, a combination of 3 components has been made:

- **NavMesh:** This component has been used to detect surfaces upon which the **NavMesh Agent** *can/cannot* walk.

- **NavMesh Agent:** This component contains all the information and constraints about Jennifer's movements (height, angular speed, acceleration, slope angle, etc...). It has been used during the *NavMesh Baking process*.
- **NavMesh Obstacles:** This component is used to represent *non-static* obstacles for the *NavMesh Agent*. It has been used for dynamic objects such as *hideout objects* (the box) and the door.

7.3.3 Scripts

MomAI.cs

This script is used to implement the **routine logic** already explained in section 7.3.2. Events for which this script has a callback method:

- *ps1Opened*: callback that enables the routine.

FOV.cs

Script used to implements Jennifer's **Field Of View**. Events raised by this script:

- *gameOver*: event raised when the player is detected inside the *Field Of View* while he is in *Alive mode* and he is not occluded by other objects.

7.3.4 Animator

A very simple state machine was used for Andy's mother animator as there are only two animations used as previously introduced. The figure (Figure 7.24) below shows the structure of the state machine used to manage transactions between the different animations.

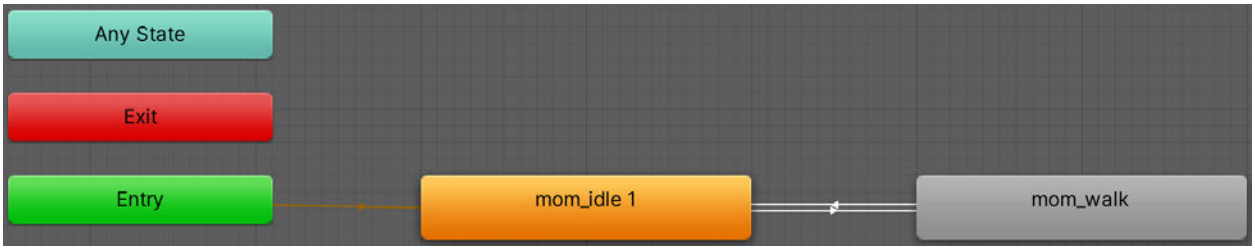


Figure 7.24: Basic state machine used for the animator of Andy's mother.

7.4 Alien

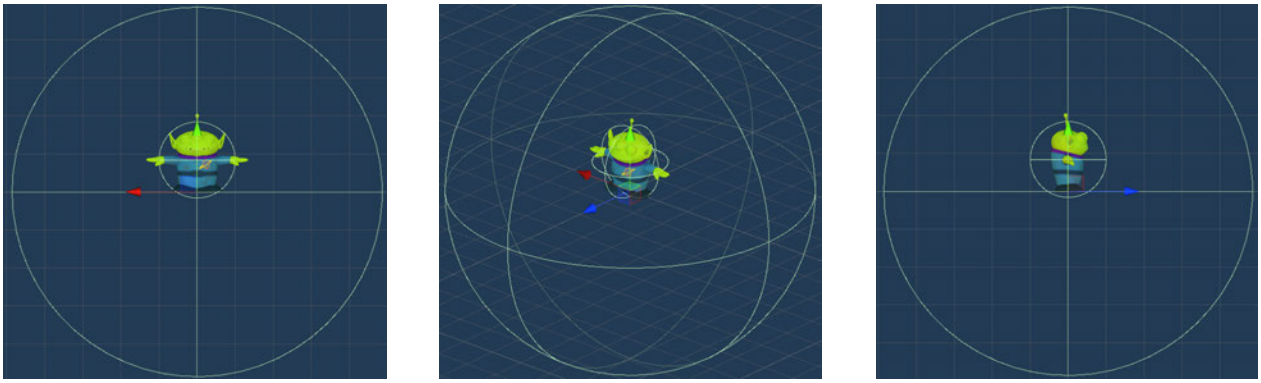
The structure of this section is the following:

- **Prefabs**: the prefab related to Aliens;
- **AI**: explanation of the artificial intelligence implemented for Aliens.
- **Scripts**: all scripts associated to Aliens to attribute behaviours to it;
- **Animator**: the animator used to associate the different animations introduced to Aliens;

7.4.1 Prefabs

Two prefab has been made for Alien: **ALIEN** and **ALIENDancer**.

ALIEN



(a) front (b) 45° (c) side

Figure 7.28: ALIEN's isometric visualization. We can notice the external trigger and the internal collider.

ALIEN's prefab has the structure shown in the picture below:

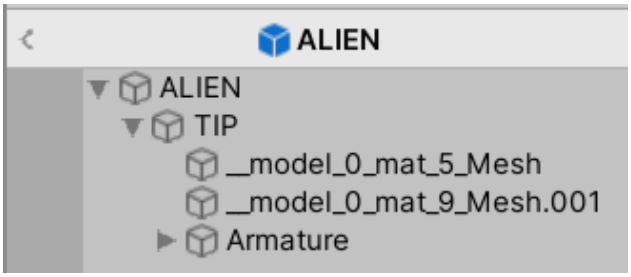


Figure 7.29: ALIEN's structure

The root object (**ALIEN**) contains 2 components:

- **Trigger collider**: used to trigger the "*hello*" animation in order to keep player's attention.
- **Alien Controller (Script)**

The second object in the hierarchy is named **TIP**. This name must be changed based on the tip that the alien instantiated owns. This object contains:

- **Animator**.
- **Collider**.
- **Outline (Script)**.

Later on, we can notice the **Meshes** that contain all the materials related to the model and the **Armature** representing the skeleton (without colliders).

ALIENDancer

This prefab is the same as the original model with only Armature (without colliders) and Meshes. The only addition is an Animator to make it dance in the end credits.

7.4.2 AI

The **artificial intelligence** of the alien provides the activation of the "*hello*" animation when the player enters the external trigger.
Then, when the player interacts with it, it comes back to "*idle*" animation and is responsible to update the text relative to the **Tip Dialog** that will be shown.
When the player exits the trigger, it comes back to "*idle*" animation.

Scripts

AlienController.cs

This script contains the logic for the alien's AI (see 7.4.2). Events for which this script has a callback method:

- *interactWithAlien*: callback that updates information inside AlienSharedInfo about the alien with which the player is interacting and raises the event *tipEnabled*.

Events raised by this script:

- *alienInteraction*: event that activate the right logic when the player goes in/out from the trigger.
- *tipEnabled*: event raised when the player interact with the alien and the event *interactWithAlien* is raised.

AliensManager.cs

This script is attached to the game object that contains all the aliens. It coordinates common logic to all aliens.

SFXAliensController.cs

Script that manages the activation of the right alien's audio clip. Events for which this script has a callback method:

- *alienInteraction* raised in *GameManager.cs*, in *UIInputHandler.cs*, in *Interaction-Mager.cs* and in *AlienController.cs*: callback that plays the right audio clip based on the parameter passed when the event is raised.

AlienSharedInfo.cs

The class defined in this script is a *Scriptable Object* and contains information about **aliens current state**. An instance of this class works like a **global reference**. It is used to share, among different classes, information about all the aliens in the scene.

7.4.3 Animator

A very simple state machine was used for Alien's animator as there are only two animations used as previously introduced. The figure (Figure 7.30) below shows the structure of the state machine used to manage transactions between the different animations.

Figure 7.30: Basic state machine used for the animator of Alien.

7.5 Interactable Objects

These are all the object that are outlined and with which the player can interact by selecting them uisng the pointer in the middle of the screen. These object are:

- **Generic Interactable Objects**: *ps1_controller* is the only one of this kind in this demo.

- **Aliens** (already explained above).
- **Hideouts objects.**
- **Collectibles objects.**

All these kind of objects have a script in common: "**Outline.cs**".

Outline.cs

This script is used in every to change properties of the material in order to outline borders with a custom color.

7.5.1 Hideouts objects

These are the object where woody can hide himself in. They have the tag containing the string "*Selectable*" and the layer "*Hideout*". The only hideout presents in this demo is the **box**. It has attached the "**Outline.cs**" script described above and the "**HideoutController.cs**" (initially disabled)

7.5.2 Collectibles Objects

All these objects have a tag "*Selectable*" and a layer "*Collectibles*" and a script attached: "**Outline.cs**". In the game object that owns all the collectibles, an **AudioSource** component and a script named "**VFX_SFXCollectiblesController.cs**" has been attached. This script defines a class that allows to manage:

- the reproduction of the speach of a collectible once Woody interacts with it;
- the reproduction of the sound related to the collection;
- the instantiation of a particle once the collection takes place.

In this script some AudioEvents are used in order to play the audio clips associated with them. These are:

- *JessieSpeach*, used to reproduce Jessie's *audio clip*;
- *BuzzSpeach*, used to reproduce Buzz Lightyear's *audio clip*;
- *RcSpeach*, used to reproduce RC's *audio clip*
- *RexSpeach*, used to reproduce Rex's *audio clip*;
- *SlinkyDogSpeach*, used to reproduce Slinky Dog's *audio clip*;
- *HammSpeach*, used to reproduce Hamm's *audio clip*;
- *BullseyeSpeach*, used to reproduce Bulls Eye's *audio clip*;

- *SmokePoof*, used to reproduce SmokePoof's *audio clip*;

Moreover, inside this script some callback methods related to the following events have been defined:

- *InteractWithCollectibles*: used for the reproduction of audio clips related to collectibles;
- *OnCollect* : used for the reproduction of the *audio clip* related to the collection of the object and the instantiation of the related *particle*;

7.6 Animated Objects

Animated Objects are objects with which the player can interact with but that are not "*Interactable Objects*" (i.e. they are not selectable by the player and are not outlined when player points them with the pointer). These objects are:

- Door.
- TV.
- Ball.

7.6.1 Door

This object is the part of the door that actually opens when mom comes. It has a script attached named "**DoorManager.cs**".
This script manages all things related to the door, like the rotation of the game object when mom comes into the room.
Events for which this script has a callback method:

- *ps1Opened*: callback that waits for some seconds and then rotate the door game object to let the mom enter in the room.

Events raised by this script:

- *ThoughtTriggered:* raised once Woody enters the door's trigger collider before the door is opened.

7.6.2 TV

This object, named "**crt**" in scene hierarchy, has several children that, togheter, forms the television in its entirety. In the children object we can find the **AudioSource** component of the speaker, used to reproduce tv sounds, and the **VideoPlayer** component of the screen plane, used to reprocuce the videoclips. The root object has a script attached to it, "**TVManager.cs**".
This script is responsible to play the right videoclip in the right moment.
Events for which this script has a callback method:

- *openPSOneEvent*: callback that plays the PSOne videoclip.
- *pause* and *resume*: callbacks that pause/resume respectively the current videoclip.

7.6.3 Ball

This object, named "**luxo_ball**" in scene hierarchy, is a simple object with just a **SphereCollider** (with a custom "*BallBounce*" physics material) and a **Rigidbody** to let it collides with other colliders in the scene.

7.7 Camera

This section is divided into 4 parts: **Game camera**, **Intro Stuff**, **Camera Credits** and **Scripts**.

7.7.1 Game camera

This subsection includes all the things related to the cameras used during all the interaction of the player related to the gameplay. The game object that encapsulates everything is "**===CAMERA===**" and its structure is shown in the figure below.

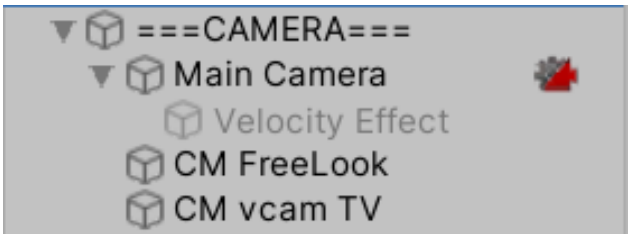


Figure 7.31: "*===CAMERA===*"'s structure.

Root: the root is an empty game object with just 1 script attached: **CameraManager.cs**.

Main Camera: This game object contains the *Camera* component used by Unity to render what the player actually sees. This object contains:

- a **CinemachineBrain** component used to manage the various *Cinemachine virtual cameras* (go here for further info).
- a script named **HighSpeedFX.cs**
- a child object **Velocity Effect** (normally disabled) that contains the Particle System componet implementing the high speed effect shown **here**. It has been placed as child of the main camera in order to make this effect to stay always in front of the camera, following its movements.

CM FreeLook: This is the object used to implement the actual Third-Person-Camera controlled by the player. It contains the following components:

- *CinemachineFreeLook*: virtual camera used to implement a Third-Person prespective.
- *Cinemachine Collider*: used to make the camera collides against colliders to prevent camera penetration.
- *Cinemachine Camera Offset*: to implement a better offset.
- *Cinemachine Free Look Zoom* (Script).
- *Target Transparency* (Script).
- *Camera Shake Controller* (Script).

CM vcam TV: This object contains *CinemachineVirtualCamera* component used to focus the view on the Television when the player switches on the PS1.

7.7.2 Intro Stuff

This subsection includes all the things related to the intro animation of the camera. The object to be considered is "**===INTRO STUFF===**".

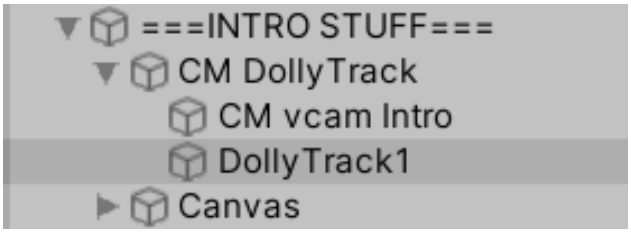


Figure 7.32: "*===INTRO STUFF===*"'s structure.

Root: The root is a simple empty object used as container for a better division of the objects in the Hierarchy window.

CM DollyTrack This is an empty object such as the root. It contains objects used to implement the camera animation that can be seen at the beginning of the level.

CM vcam Intro: This object contains the following components:

- *CinemachineVirtualCamera*
- *Animator* and *Playable Director*: these objects, together with a timeline asset named "*Timeline Intro*", menages camera animations.
- *Time Scale Changer* (Script).

DollyTrack1: This object contains the component *CinemachineSmoothPath*, provided by Cinemachine. It consists of 25 waypoints, each of which has been accurately positioned to offer a global view of the room, that form the track on which the camera moves in the intro.

Figure 7.33: In green the Track used by camera to move on the scene in the level intro.

Canvas: This object is a canvas used to display the texts and the logo visible in the intro.

7.7.3 Camera Credits

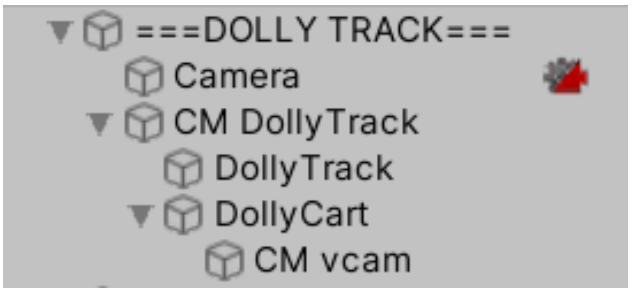


Figure 7.34: "===DOLLY TRACK===''s structure.

The figure above shows the structure of the object used to implement the Dolly Track camera provided by Cinemachine and customized to fit our purposes.

Root: Empty object that acts as container.

Camera: object that contains the *Camera* and the *CinemachineBrain*.

CM Dolly Track: Empty object that acts as container.

DollyTrack: As for Intro Stuff, this object contains the *CinemachineSmoothPath* component with 13 waypoints that form the track on which the camera moves in the end credits animation (see Fig.7.35).

DollyCart: Object that contains the *Cinemachine Dolly Cart* component provided by Cinemachine that is used to constraint the transform of the main camera to stay stuck on the track.

CM vcam: This object contains the following components:

- *CinemachineVirtualCamera*: this virtual camera uses a *custom noise profile* named "CustomShake" to enhance the "disco" effect of the credits animation.
- *Animator* and *Playable Director*: these objects, together with a timeline asset named "*Timeline Dance*", menages camera animations seen in the end credits.
- *Camera Win Loop* (Script).
- *Camera Win Scene* (Script).

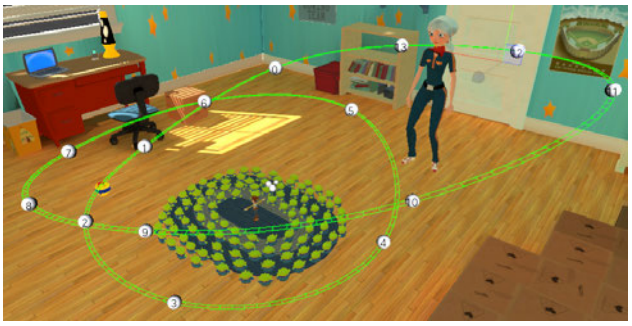


Figure 7.35: "===DOLLY TRACK==="'s structure.

7.7.4 Scripts

CameraManager.cs

This script manages activation/deactivation and the switch between the 2 virtual cameras used during gameplay (FreeLook and TV cameras).
Events for which this script has a callback method:

- *tipEnabled* and *tipDisabled*: callbacks that disable/enable respectively the FreeLook virtual camera that owns camera controls.
- *ps1Opened*: callback that switches the virtual camera rendered (from *FreeLook* to *cameraTV* and, then, from *cameraTV* to *FreeLook*) for a certain number of seconds.

HighSpeedFX.cs

Script used to activate/deactivate the object that contains the Particle System relative to the *High Speed FX* (**here** the effect).
Events for which this script has a callback method:

- *startHighSpeedFX* and *stopHighSpeedFX*: callbacks that activate/deactivate respectively the *High Speed FX* object.

CinemachineFreeLookZoom.cs

Script that implements InputActions.ILookControlsActions interface in order to manages player inputs relative to zoom changes.

TargetTransparency.cs

Script used to implement a *trasparency* effect to the material of the object the camera is following (i.e. the player's model, woody). This effect is applied when the camera is too close to the player and the model prevents to see what's ahead.

CameraShakeController.cs

Method used to make the camera shakes when player velocity is very high. Events for which this script has a callback method:

- *startCamShake* and *stopCamShake*: callbacks that activate/deactivate respectively the camera shaking.

TimeScaleChanger.cs

This script is used to change the **Time.TimeScale** of the game in order to create a sort of *slow motion* for the level's intro animation.

CameraWinLoop.cs

This script is used to constraint the end credits camera to move on the track with a constant velocity. Without this script, the camera was moving at different speeds depending on the distance between 2 waypoints.

CameraWinScene.cs

This script is used to implement a change of camera *LookAt* between Jennifer and Woody during end credits scene.

7.8 UI

The UI has been implemented to be responsive. For groups of elements that had to be displayed horizontally (like UI collectibles coins [see Fig.7.31]), *Horizontal Layout Groups*

have been used. Same for vertical groups of elements (like menu buttons [see Fig.7.32]), where *Vertical Layout Groups* have been used.
For the Dialog Boxes (see here), Sprite Editor has been used in order to obtain a resizable Dialog Box that adapts its dimensions based on the content size.
Moreover, the various elements of the UI have been anchored in a reasonable way, in order to maintain approximately the same structure when resolution changes.
The following section defines the scripts related to the management of the components within the UI.

7.8.1 Scripts

Three scripts related to the UI have been defined:

- UICollectiblesManager.cs;
- IngameUI.cs;
- UIInputHandler.cs.

UICollectiblesManager.cs

This script defines the UIControllerManager class. This class allows to manage the activation and the different animations of the UI Images related to collectibles. In particular, the animation of each single UI Image is divided into three main phases:

- Rotate: a rotation is attributed to the UI Image;
- ScaleUP: the size of the UI Image is increased;
- ScaleDown: the size of the UI Image is decreased.

Events for which this class has a callback method:

- *OnCollect*: callback that defines the animations of the images relating to the collectibles using the methods used for the phases described above.

IngameUI.cs

This script defines the InGameUI class, which allows to manage all operations related to the user interface while the game menu is shown. In particular, several operations concerning the start menu, the pause menu, the game over screen have been managed:

- initialization of the elements composing the menu (including those related to the settings);
- activation and deactivation of the items composing the menu;
- management of actions related to the menu;

- editing and reproduction of the *audio clip*.

Moreover, it manages the initialization and editing of the resolution at which the game is executed. The class also defines some callback methods related to the events:

- *Pause*, callback that defines:
  - the transition between the background music used during the game and that used for the menu;
  - the activation of objects related to the menu;
  - the visualization and unlocking of the mouse.
- *GameOver*: callback used to enable the game over canvas.

Events raised by this class:

- *Resume*: raised during the transition from pause mode to in game mode. It is used to:
  - stop the AudioSource related to the background music for the menu;
  - play the AudioSource related to the background music used in game;
  - deactivate the canvas related to the menu;
  - the deactivation and the locking of the mouse in the centre of the screen.

UIInputHandler.cs

This script defines the UIInputHandler class that allows to:

- enable or disable dialogs related to Alien Tips and Woody's thoughts;
- manage the transitions between the various alien tips within the related dialog;

Events for which this class has a callback method:

- *tipEnabled*: callback used to show the dialog related to the alien tip together with the definition the transition from one tip message to the next one;
- *thoughtTriggered*: callback used to show the thought dialog together with the thought message;

Events raised from this class:

- *AlienInteraction*: this is raised each time the player interacts with the alien in order to move on to the next tip;
- *TipDisabled*: this is launched once the alien has finished the tips, so there are no more tips to show within the dialogue;

7.9 EXTRA

7.9.1 Game Manager

This is the object responsible for game flow management and for the overall proper functioning of the game. It is a simple empty object that owns a sript named GameManager.cs.

7.9.2 Lights

To implement the lighting of this game, both static and dynamic lights have been used. Going into detail, there are 4 light components:

- **Directional Light:** used to simulate sun light.
- **2 Area Lights:** lights placed outside the 2 windows, used for e better lighting of the room interior.
- **Light Prob Group:** used to better simulate light that bounces on objects.

(a) Top Down view (b) Side view (c) 45° view

Figure 7.39: Light Probs Group's isometric visualizations.

7.9.3 Scripts

GameManager.cs

It is in charge of instantiating the prefab *WOODYAlive*, of managing the transition from the level intro to the effective game start, of loading the end credits scene when the player win the game.
Events for which this script has a callback method:

- *ps1Opened*: callback that sets the relative boolean value inside GameSharedInfo.cs to **true**.
- *gameOver*: callback that sets the relative boolean value inside GameSharedInfo.cs to **true** and sets the *TimeScale* to 0 in orther to stop the game.

- *resume*: callback that restores the variables responsible for the correct resume of the game.
- *win*: callback that loads the end credits scene.

Events raised by this script:

- *interactWithAlien* and *alienInteraction*: raised when the level intro finishes and the alien dialog must be shown up.
- *pause*: raised when player goes to the menu.

GameSharedInfo.cs

This script allows to share all informations related to the game manager and it also allows to initialize them with their default values.

Feature Works

As mentioned several times in previous chapters, this was initially intended to be a bigger project. Many features have been discarded in the development of this demo and many things can be improved. Some of the ideas that have been left for future implementations are explained below.

8.1 Upgrades

8.1.1 Woody's upgrades

At the moment, whenever woody climbs a surface, it doesn't matter if there is a wall in front of woody's pelvis or not. Woody will perform always the same "climb" animation that, in case of wall ahead, can cause the legs to climb an "*invisible*" wall. There is the need to import just one new animation in order to choose the right one depending on the already implemented "*wall climb check*".

8.1.2 Graphics upgrades

First, **shadows** and **lights** need to be *polished*. After that, the idea was to make the look & feel more cartoonish by using some **post-processing fx**, increasing color saturation, and by using *toony* **shaders**.

8.2 Levels

The original idea was to have the *entire house* available to be explored, with initially locked areas that the player could unlock later on. At each "level", player can unlock the next one by fully exploring the current one and by solving some puzzle. Other levels were planned, among which can be mentioned **Bathroom, Corridor & stairs, Kitchen** and **Living room**.

8.3 New Game Mechanics

A key mechanic, inspired by Grand Theft Auto 5 , that was intended to be developed was the possibility to switch between 2 charachters: **Woody** and **Buzz Lightyear**.
Woody had to be able to use his **lasso** (in order to cross ditches) and to ride his friend **Bullseye** or **RC Car** (to quickly travel across large areas in the house).
Instead, by impersonating Buzz Lightyear, player could use its ability to **fly** and to **shoot laser** in order to solve some puzzle that requires these skills.

8.4 Andy's Mother

At the current state of development, **Mom's AI** is very simple but the original idea was to implement more complex behaviours. Jennefer had to follow a routine that would make her explore the various areas of the house, implementing a behaviour similar to the one used in Alien: Isolation. Moreover, also sound and visual effects need to be implemented.

8.5 Bugfix & Refactoring

Obviously, some bugs are present in this demo and they have to be fixed in future work. One of the most annoying bug is that when the player resumes the game from the menu, the in-game music and the video on TV are played again from the beginning instead to be resumed. Another bug is that the interaction of the player with the PSOne, when it is already switched on, will cause the popup of the woody's thought. Moreover, the player is able to go in *Ragdoll* mode during the opening of the PSOne.

8.5.1 Refactoring

When crunch time came and we realized that the time was running out and there was a lot to do in order to finish this demo, we had to implement many things without thinking too much about how to do them in the most efficient/effective way. This means there is a lot that can be improved.

Event System: this is one of the most messy parts and, consequently, one of those that can be improved the most. It can be refactored in order to implement the **Observer Pattern** in a more accurate way, creating a "*Messanger*" class that owns all the game events and, at the same time, centralizes the subscription of the callback methods to the various events and the broadcasting of the events raised. In this way, each script that must use events, could have just a reference to the Messenger class, instead of having a reference to every single event it uses.