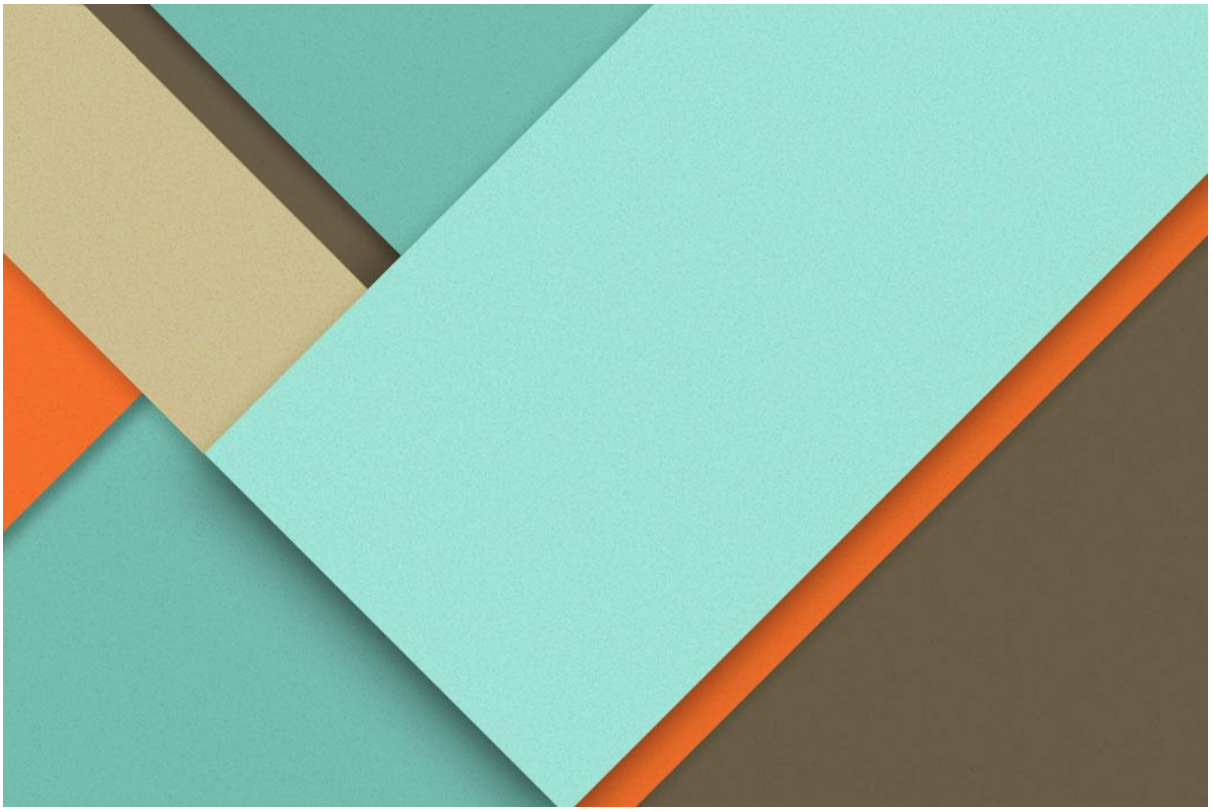


Engine Architecture and Design

Technical Document - Protoype “Dead Sapling”

Florian Federhofer - ai21m019@technikum-wien.at
Martin Gritsch - ai21m020@technikum-wien.at
Christina Obereigner - ai21m006@technikum-wien.at
Jonas Wendel - ai21m007@technikum-wien.at
Sünje Winteler - ai21m038@technikum-wien.at

Content

Team

The team consists of:
Florian Federhofer (ai21m019@technikum-wien.at)
Martin Gritsch (ai21m020@technikum-wien.at)
Christina Obereigner (ai21m006@technikum-wien.at)
Jonas Wendel (ai21m007@technikum-wien.at)
Sünje Winteler (ai21m038@technikum-wien.at)

Goal

The goal was to develop a basic prototype for the development project for the Team IceCream Connection (Martin, Florian, Jonas and Sünje) with additional help from Christina. The project “Dead Sapling” is supposed to be a 3rd person tower building game with an intricate story, set up in a cyberpunk-hippie universe.
The set goals for the first semester were basic variants of the following game aspects:

- Enemy Units (including AI)
- Towers
- Map
- Buildings
- Fighting
- Player-World interaction (Movement, Animations, etc..)
- Illumination
- Sound
- Particle Systems
- UI / Menu

The goals were (partially) met and their implementation is described in the following chapters.

Development Methodology

As the project and its blueprints grew larger the decision was made to implement larger and self-contained functionality blocks in C++ and manage the “flow” via exposed variables & functions in the corresponding blueprints.
This enabled us to have better oversight, debug abilities and a more readable codebase.

Key Bindings

Action / Movement	Binding
MoveForward	W, S, Up, Down Gamepad Left Thumbstick Y-Axis
MoveRight	A, D Gamepad Left Thumbstick X-Axis
Turn	Mouse X Gamepad Right Thumbstick X-Axis
Look Up	Mouse Y Gamepad Right Thumbstick Y-Axis
Jump	Space Bar Gamepad Face Button Bottom
Interact	Left Mouse Button Gamepad Face Button Right
ToggleBuildMode	B Gamepad D-pad Up
ExitBuildMode	Escape, Right Mouse Button Gamepad D-pad Down
SlamDown	Left Ctrl, Right Ctrl
Thrust	Left Shift, Right Shift Gamepad Right Trigger Axis
Overview	C Gamepad Left Trigger Axis
StartNight	T Gamepad Face Button Top
MainMenu	Escape Gamepad Special Right
Run	Q Press Left Thumbstick

Abbreviations

Word / Meaning	Abbreviation
Blueprint	BP
Blackboard	BB
Behavior Tree	BT
Behavior Tree Task	BTT
Player Character	PC

Development Overview

The following parts are in no particular order and are supposed to provide an overview of the state of the project. This is not an extensive overview, the minor changes some classes won't be described in detail e.g.:
GameMode - added custom classes
GameInstance - used to provide static tower data
TowerInfo - DataAsset for information about towers

Enemy Units (including AI)

Enemy units currently exist only in one form and are spawned after a short countdown on BeginPlay. They have a pathfinding, which is currently randomly selecting a path until they figure out how to get to the "End" point at which they take damage and are despawned.

BP_AI_Character

Provides a Mesh, Hitbox, Healthpoints and functionality for taking Damage & informing the corresponding Group of its death.

BP_AI_Controller

Just runs the corresponding behavior tree on spawn (=BeginPlay-Event).

BP_AI_Group_Pawn

Used as formation mechanism by providing positons for each member of the group to follow. This allows for a more ordered and uniform walking style - which was required due to the enemies being spawned by an AI according to the world setting / storyline (which wouldn't just spawn a horde but a more organized formation of enemies)

BP_AI_Spawner

Spawns BP_AI_Group_Pawns.

BP_Group_Manager_Controller

Just runs the corresponding behavior tree on spawn (=BeginPlay-Event).

BB_Enemy_Blackboard

This is a basic Blackboard for the Enemy Pawn, currently only consisting of just the corresponding walking target in the GroupManager.

BB_GroupManager_Blackboard

Holds the NextWayPoint to go to for the Group.

BT_Enemies

Currently just calls the BTT_MoveEnemies.
The Plan: There will be added checks to see if the PC is in Range of the BP_GroupManager(this is the max. range they will follow the PC) to be attacked & then attack it.

BT_GroupManager

Currently it just calls BTT_GetNextWaypoint, moves to the retrieved waypoint and waits for a short time.
The Plan: There will be added checks to see if the PC is in Range of a BP_AI_Character and the stop movement until it isn't - the BP_AI_Character shouldn't follow their GroupManager until the PC is dead / not in Range of the BP_GroupManager anymore, if the PC is not in Range/Alive.

RoutePoint - C++

Currently just a data class about the neighboring routes that are possible and the ABuildingSpots that are in its range.
The Blueprint version spawns a BP_PathLight to show the player where the enemy will be walking.
The Plan: This will be used to calculate weights for each route according to the buildings built.

DeadSaplingGameState - C++

The gamestate will be used to calculate a new path for the enemy AI. This is currently just a randomized path - here we will implement the A* algorithm.

Interactions

The C++ InteractiveActor-Interface allows us to trace the PCs view and interact with traced objects, this currently only supports ABuildingSpots, but will be extended for NPCs, Items and other interactable objects.

InteractiveActor - C++

Provides a plain interface guaranteeing the implementation for Interact & OnTrace functions.

BuildingSpot - C++

Provides a mesh that indicates to the player where the building can be built in the world. When the build mode is active and a buildingspot is traced it will display the selected tower - to preview the tower and in the future the stats of the tower.

Tower - C++

Base class for Towers, copies data from the provided DA_TowerInfo and sets the mesh / stats accordingly.

Map

The map has been built with the world setting of an enemy AI & a “friendly” AI defending itself - by abusing the player to do so. So it shows the village the player has to defend and a canyon construct that has similarities to the graph of a neural network. A blending landscape materials have been used and a river with wave simulation has been added.

Water

The river was made by following a tutorial for stylized water materials and putting this material on a plane.

Foliage

The map was decorated in the foliage mode with different stone, grass and wheat assets.

Illumination

Two post processing volumes were used for changing illumination and the color mood of the game depending on the location of the player (farm or battlefield). The game (currently) plays during noon/night time and is illuminated via light sources such as light poles and lanterns.

Farm

The post processing volume of the farm aims to give the player a welcoming atmosphere. The color grading is set to a lighter greenish/blueish tone. The Lense Flare has a very low intensity and is rather blurry to make the player feel more welcome.

Battlefield

When moving from the farm to the battlefield the colors change to a darker and more aggressive orange tone with a sharper and more noticable Lense Flare. Additionally a light Vignette Image Effect is used to make the player more focused.

The paths of the battlefield are illuminated with lanterns. The lanterns use a Point Light to light up the surrounding area. Additionally the Blueprint LightFlickering was created (based on a tutorial on how to create a campfire) that randomly sets the intensity of the Point Light after a short amount of time.

Fighting

Towers

Towers are currently shooting simple bullets (read flowers) on hit event deletes the bullet and does damage to the enemy, for this the following are involved:
BP_AI_Character - checks on hit and takes 100 damage right now
BP_DefaultTower - Spawning Ammunition to shoot at enemies entering it’s capsule sphere
BP_BaseAmmo & BP_FlowerAmmo - as Ammunition to be spawned

Player-World Movement / Camera

Player-Movement

The player has certain types of movement, which can be used to traverse across the map or to get an overview of where the enemies are.

Walk - the player can walk over the map. Walking uses the UE5 standard implementation. The walking direction corresponds to the direction of the camera and which direction is pressed on the left thumbstick.

Run - running is basically the same as walking, just faster. It's a simple Blueprint where an event action (Q or Left Thumbstick is pressed) is triggered and the walkspeed is increased until the button is released.

Jump - the player has the ability to jump two times. This is controlled with a "timesJumped" variable.

Rocket Boost - a timeline is triggered which gives the player upwards thrust and reduces the fuel the player has. If the fuel is empty the player falls back to the ground. Fuel is refueling when the player stands on the ground.

Overview - this is a rather complicated Blueprint which eventually should be rewritten in C++ for more simplicity. But basically the player triggers an event action when a button is pressed (C or right trigger button) which sets of a timeline to give the player upwards thrust. When the player has reached the highest point a new timeline is triggered to smoothly widen the

field of view to give the player a better overview over the map. Also the models are hidden in this state. When the button is released everything is done in reverse.

Ballon

The balloon only appears when the player is using Overview or Thrust.

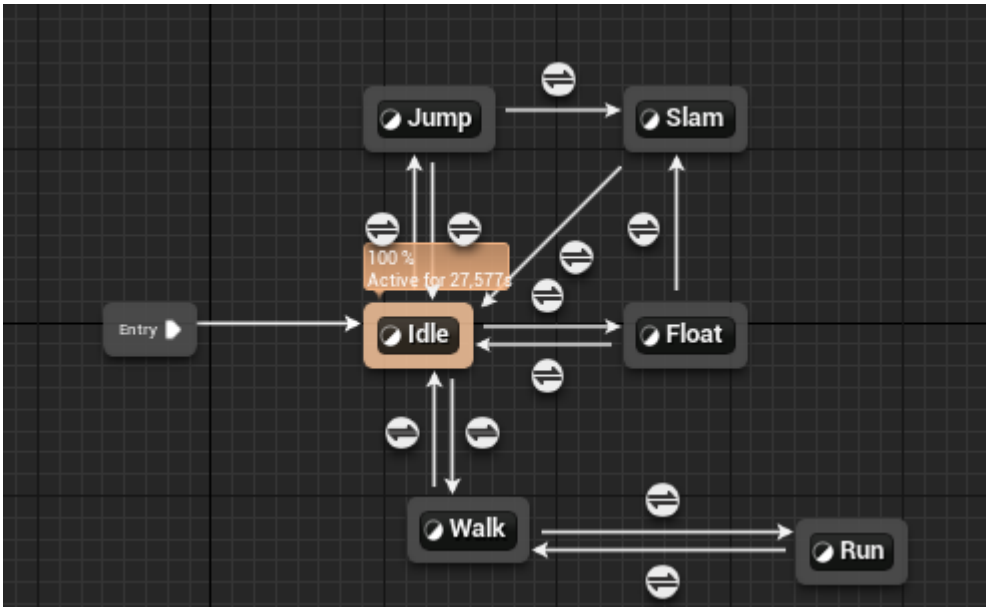
Animation

Player Animation

The animation of the player is specified in the Player_AnimBP and the Ballon_AnimBP for the player movement animation and balloon animation, respectively.

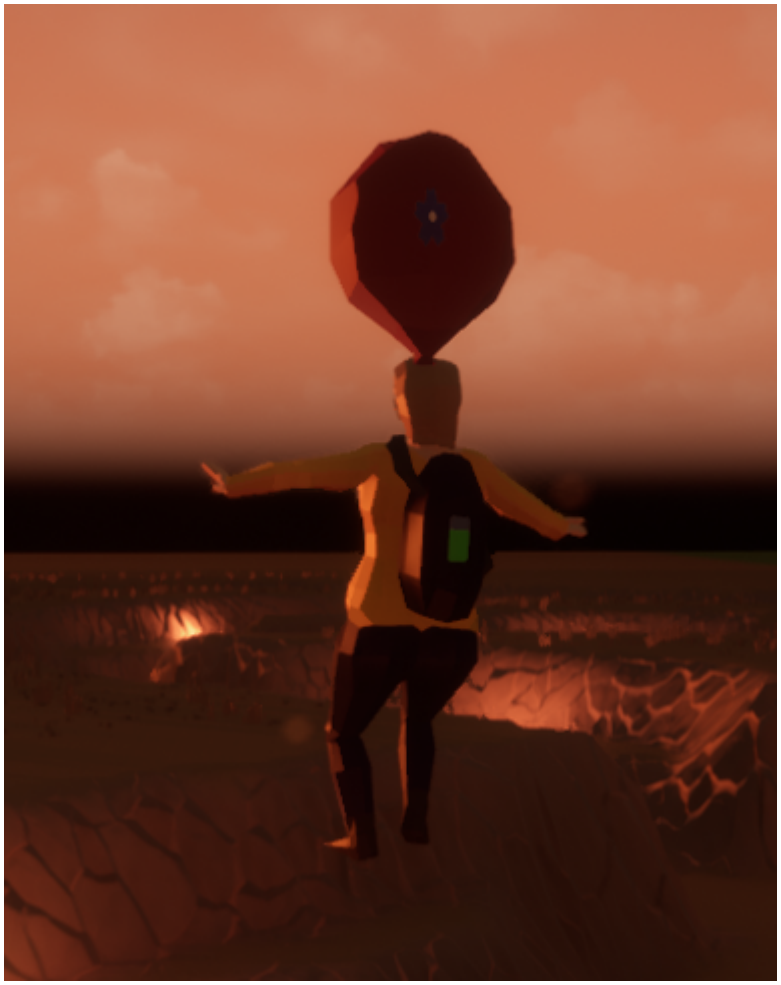
Player Movement

There are six states of animation for the player movement.
In idle the animation shows the player standing around. This animation is called when the player is on the ground and not moving. This is the default state.
The walk state is called when the player is on the ground and moving.
The run state can only be called when the player is walking. It is called when the run button is pressed. Consequently when the player stops pressing the run button the walking animation is played again.
The float state is called when the player is in the air using either Thrust or Overview. It is used with both the Is Flying and the Is Falling Nav Movement Component. The float state is not called when using the Jump key.
The Jump state is called when the player uses the Jump key.
The Slam state can be accessed from the Jump state or the Float state and is called when the player presses the SlamDown button.



Ballon

The balloon uses an Animation Montage (Ballon_Montage) which is implemented in the Ballon_AnimBP.
The animation is played once when the balloon appears. The balloon is basically inflated with the gas from the tank on the back of the character.



Enemy Animation

The enemy animation has two states: idle and walk. Idle is the default state. Walk is called when the enemy is moving on the ground.

Sound

River

The river sound is directional and plays in a loop, it has a fall-off distance and its position is adapted on a BP_RiverSound_Spline to the player location which allows for an easy soundeffekt that feels somewhat realistic.

Enemies

Walking a sound for the enemy army has been added. The animation throws notifications in the frame the foot touches the floor, this is caught in the animation blueprint for the enemy.

Background Music

Background music is played dependent on the state of the game using a Switch node. During the night one music is played, when the player comes within a certain distance of the enemy, the music switches to the fight music.
Currently there are two states, Night and Fight, this can be adapted to more states (e.g. Day, Build, etc.).

Particle Systems

Path Light

A simple path light emitter shows the path of the enemy, path lights are spawned on wave setup and destroyed as soon as the enemy walks through the BP_RoutePoint hitbox.

Butterflies

A particle effect of butterflies was created for the farm area. The Niagara FX Hanging Particulates was used as a template and then altered to spawn and despawn butterflies in an area that move with a random linear force. The butterflies are animated with a 2D sprite sheet that is then played via a Sub UVAnimation during Particle Update.

Lantern

Each lantern that guides the player the way has a simple fire and smoke particle effect attached to it. This effect was taken from the POLYGON asset packs.

Assets

Asset packs of the POLYGON series by Synty Store were used, as well as music by Dark Fantasy Studio.

The following assets were created by us:
Map
Player Character
Butterflies

Lessons Learned

1. Scope: The scope of the game was gigantic in the beginning and in some parts still is. We did scale it down gradually but still got caught up in seemingly important stuff that wasted days of work and got trashed in the end because it didn't work out with the bigger vision/the continuing development. The lesson we learned: keep your scope small and iterate if you got stuff done, don't take on a giant if you have problems fighting a snail.
2. Prototyping: A lot of work was done that got thrown out after the next meetup with our supporting professor. This enforced a fast paced heavy prototyping based trial and error development approach.
3. Design for the player: In the beginning our mindset and thus the world we built was heavily influenced by the ideas we had about our game from games we played and the story we wanted to write. The focus moved to a heavy player centric design approach e.g.: What are abilities we want to give the player and how are we building the world around the players abilities.
4. C++ vs. Blueprints: After the first more sophisticated systems were implemented and the readability, maintainability and adaptability of the produced blueprints went downhill fast the conscious decision was made to use C++ and well established design patterns. Currently C++ is used to provide specific and complex functionality exposed to blueprints. Blueprints are used for coordination and flow management as well as easy access and fine tuning mechanics.