EcoSurvivor Documentation

Demo Video:Link
Download Demo:Link

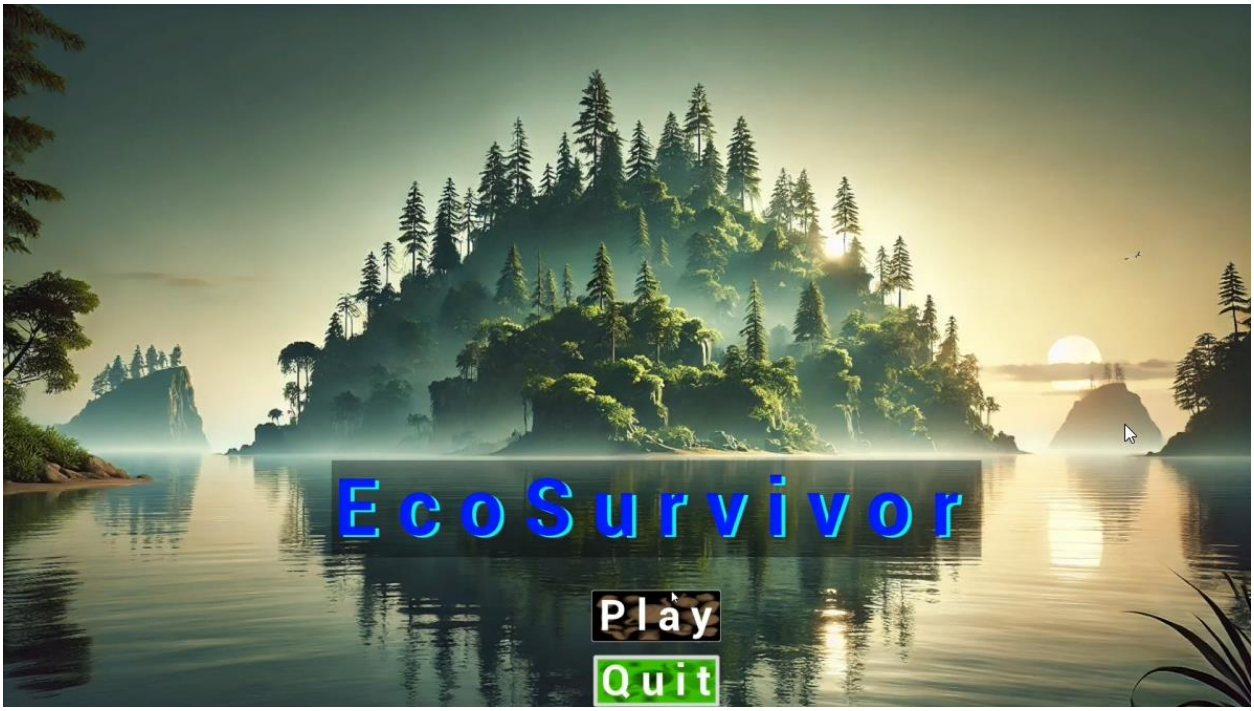
3D Game Development Using Unreal Engine 5 [Industry]

content

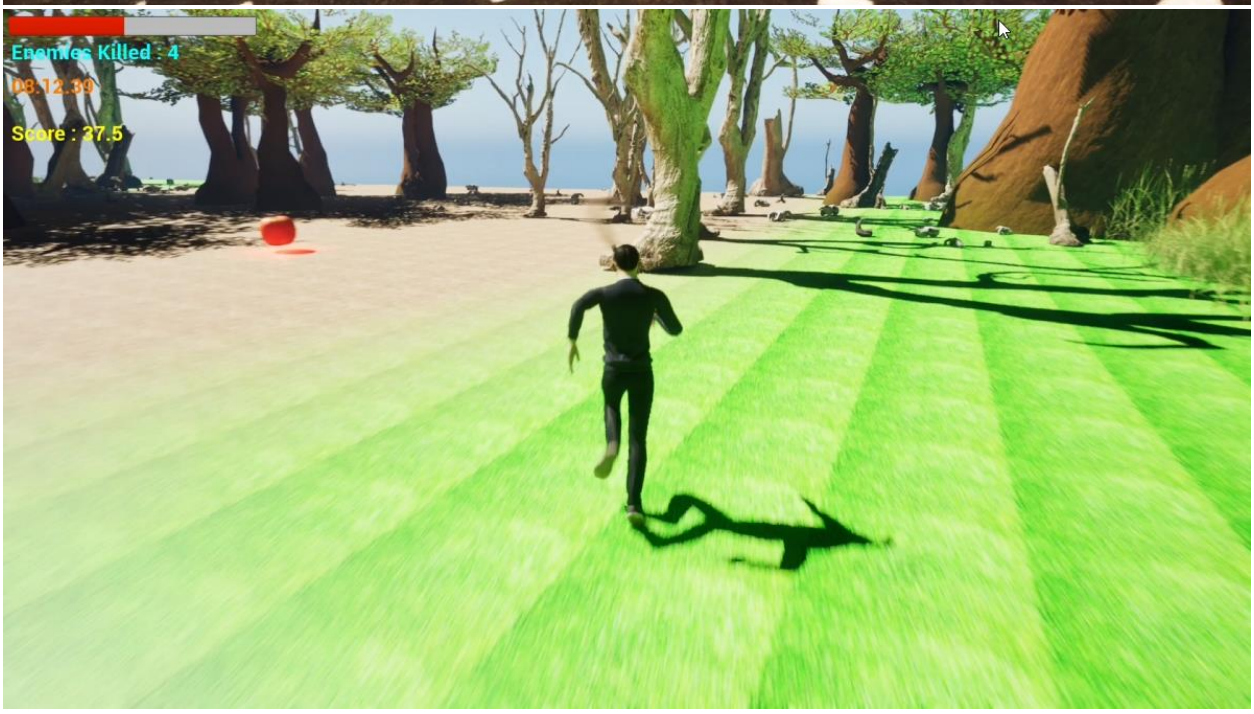
Detailed game description

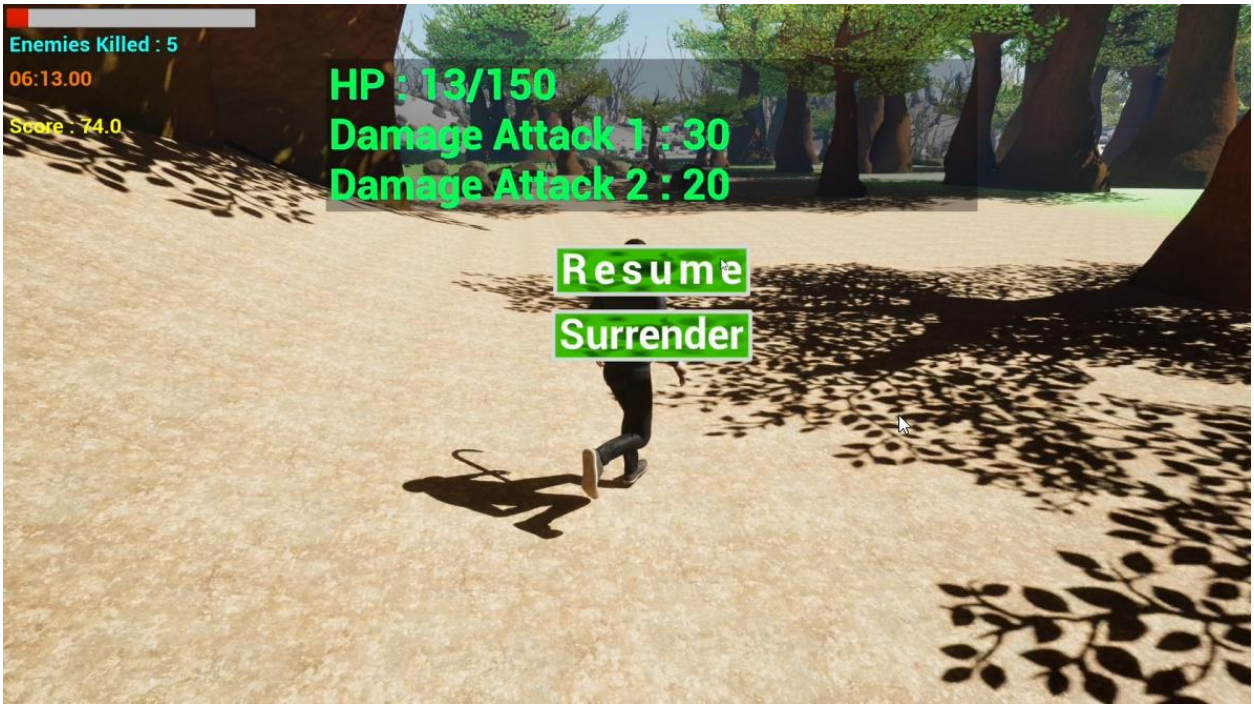
EcoSurvivor is a survival game with strategy elements, in which the player wakes up on an isolated island after a plane crash. With limited resources and surrounded by dangers, the objective is to survive long enough to find an escape route. The player can gather resources, avoid or face threats, and build a help signal. The game ends either by escaping or by death, and the final score is calculated based on survival time and the number of enemies eliminated. Inspired by titles such as *Minecraft*, *The Forest* and *Shelf*, EcoSurvivor emphasizes exploration, strategy, and the fight for survival in a hostile and unpredictable environment.

Gameplay begins with the player placed on the island, having a limited time of 10 minutes to accumulate the highest score possible. The score is influenced by the time spent alive and the number of enemies eliminated. Every 10 seconds, the island decreases the player's HP by 10, forcing them to find food for regeneration and resources to improve their attack power. A single *flare gun* is hidden on the map, and finding it is the only chance to escape. Enemies move randomly, visually searching for the player, and if the player tries to leave the island through the ocean, he is instantly eliminated. The island is not an ordinary place, but a magical and supernatural one, where the rules of nature work differently and every decision can make the difference between life and death.


EcoSurvivor
Play
Quit


Enemies Killed : 0
Score : 0.0


Enemies Killed : 4
Score : 37.5


Enemies Killed : 5
06:13.00
Score : 74.0
HP : 13/150
Damage Attack 1 : 30
Damage Attack 2 : 20
Resume
Surrender


Enemies Killed : 5
Score : 74.0

Technical specifications

EcoSurvivor was developed and optimized for the Windows operating system, offering a fluid and adapted experience for this environment. In the development version, the game occupies approximately 10 GB on disk, but the shipping version has been considerably optimized, reducing the final size to approximately 1.50 GB.

The game does not require an internet connection to function, offering a completely offline survival experience. All in-game mechanics and challenges are handled locally, without the need for external servers or online connectivity.

EcoSurvivor was developed using Unreal Engine 5.4.4, thus benefiting from the latest recent graphics and optimization technologies. This version of the game engine allowed the implementation of advanced visual effects, performance optimization and the management of a dynamic and immersive gaming environment.

Thematic specifications

EcoSurvivor falls into the survival and strategy game categories, offering an experience of exploration and combat in a hostile environment. The game's theme was chosen to combine

the strategic challenge of resource management with the unpredictability of a supernatural environment. The island, as a central element of the narrative, adds a layer of mystery and danger, transforming each game session into a battle against time and the unknown elements.

The game is single-player, focused on the individual survival experience and optimizing your own strategies. Each session lasts a maximum of 10 minutes, but due to the random generation of objects and the strength of enemies, the game can be replayed infinitely. Each run is unique, always offering new challenges and opportunities to get a higher score.

Descriptive tags: survival, strategy, exploration, resources, dangers, supernatural, island, adventure.

The game is available exclusively in English, to ensure greater accessibility. wide and easier integration into the global gaming market.

Available actions. Control.

EcoSurvivor is exclusively optimized for keyboard and mouse, providing a smooth and precise experience for players. Character control is intuitive and allows for quick movements and strategic actions while surviving on the island.

- Movement:**WASD**
- Jump:**SPACE**
- Running:**SHIFT**
- Pause:**P**
- Main attack:**Left click**
- Secondary attack:**Right click**
- Changing perspective (Third-Person / First-Person):**Mouse Wheel Press**

Players do not have the option to redefine their controls in the current version, the control being optimized for this standard scheme, ensuring a coherent and balanced experience.

Own classes

In the development of EcoSurvivor, we did not create our own classes from scratch, but rather extended existing classes from Unreal Engine. This approach allowed for efficient use of the architecture provided by the game engine, reducing development time and optimizing the integration of specific mechanics.

Algorithms

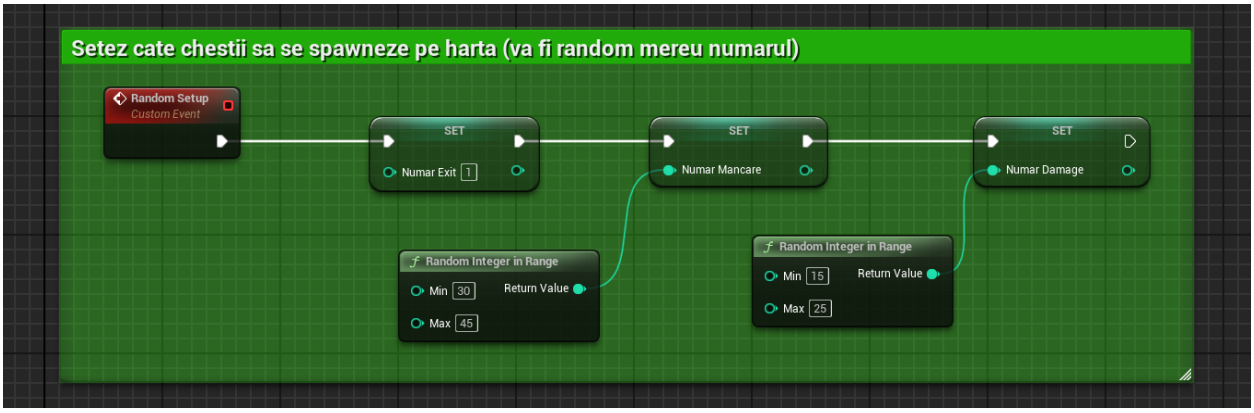
EcoSurvivor uses several algorithms to manage the dynamics of the game, one of the most interesting being the random generation of objects on the map. This algorithm ensures variability on each run, making each game session unique.

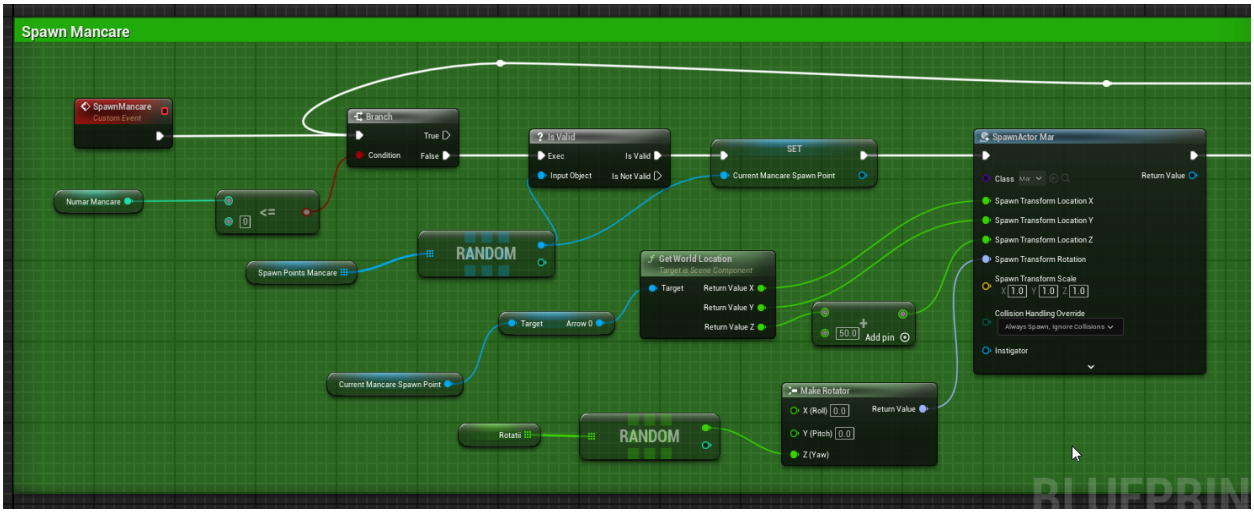
The algorithm works based on a predefined set of**spawn points**, represented in the engine by actors that contain arrow components to indicate position. Each session starts with a random number of objects to be generated based on these predefined points.

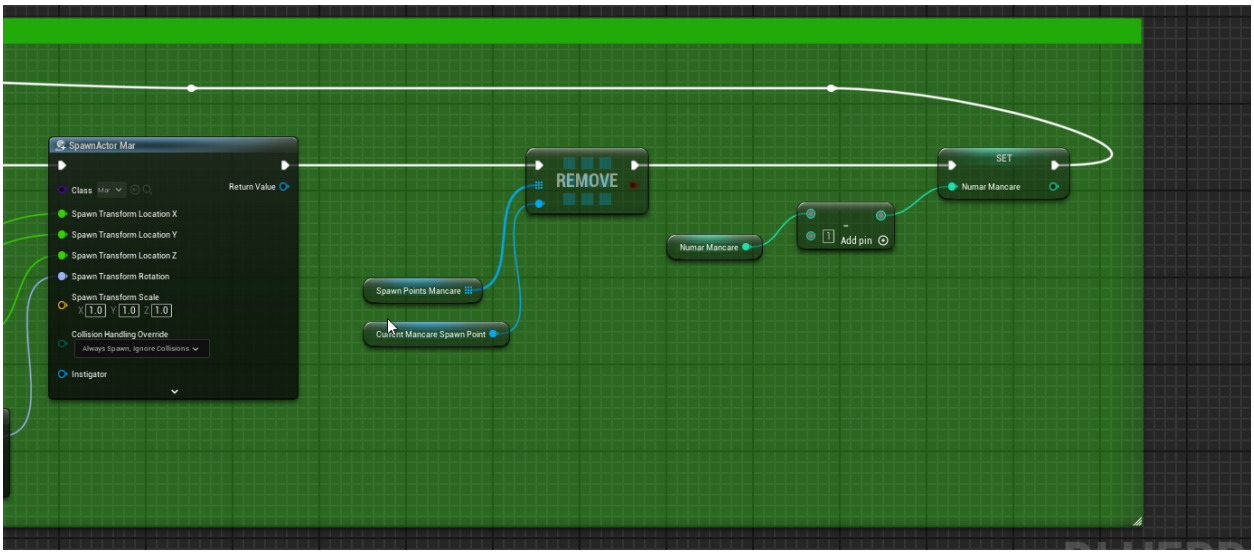
For example, let's take the case of food, which is an essential element of the game. There is a total of**60 spawn points**strategically placed on the map for food, but the actual number of items generated varies between**30 and 45**at each run. The generation procedure follows these steps:

1. A list of all available spawn points is initialized.
2. Determine the total number of objects to generate (e.g. between 30 and 45 for food).
3. Enter an iterative loop until all necessary objects have been placed.
    - A spawn point is randomly chosen from the list.
    - Place the corresponding actor at the spawn point coordinates.
    - The generated object is randomly rotated in one of the four cardinal directions (N, S, E, W).
    - The used spawn point is removed from the list to avoid its reuse in the same run.
4. The process is repeated until all necessary objects are generated.

This method ensures both variability and a high degree of optimization, avoiding overlapping objects and guaranteeing a balanced distribution on the map. The algorithm has a **time complexity O(n)**, where*n*is the total number of objects generated, since each object is placed in a single step and does not require recalculating existing positions. From a memory perspective, using a temporary list for spawn points takes up additional space of**Man)**, where*m*is the total number of initial spawn points, which is negligible compared to the available resources.


Setez cate chestii sa se spawneze pe harta (va fi random mereu numarul)
Random Setup
Custom Event
SET
Numar Exit 1
SET
Numar Mancare
SET
Numar Damage
Random Integer in Range
Min 30
Max 45
Return Value
Random Integer in Range
Min 15
Max 25
Return Value


Spawn Mancare
SpawnMancare
Custom Event
Branch
Condition
True
False
Is Valid
Exec
Input Object
Is Valid
Is Not Valid
SET
Current Mancare Spawn Point
SpawnActor Mar
Class
Return Value
Spawn Transform Location X
Spawn Transform Location Y
Spawn Transform Location Z
Spawn Transform Rotation
Spawn Transform Scale
Collision Handling Override
Instigator
Numar Mancare
<=
0
RANDOM
Spawn Points Mancare
GetWorld Location
Target is Scene Component
Target
Return Value X
Return Value Y
Return Value Z
Add pin
50.0
Target
Arrow 0
Current Mancare Spawn Point
Make Rotator
X (Roll)
Y (Pitch)
Z (Yaw)
Return Value
Rotatii
RANDOM


SpawnActor Mar
Class
Return Value
Spawn Transform Location X
Spawn Transform Location Y
Spawn Transform Location Z
Spawn Transform Rotation
Spawn Transform Scale
Collision Handling Override
Instigator
REMOVE
Spawn Points Mancare
Current Mancare Spawn Point
Numar Mancare
1
Add pin
SET
Numar Mancare

Artificial Intelligence Tools in Unreal

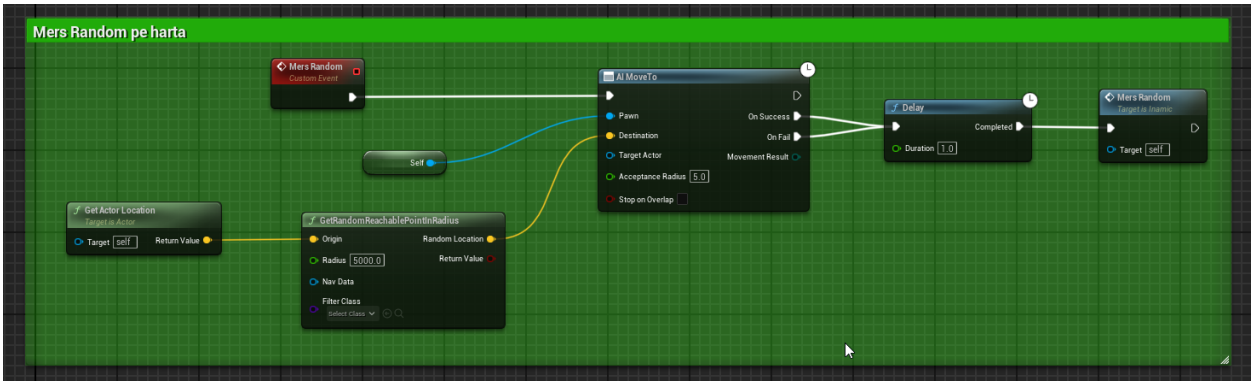
EcoSurvivor uses several artificial intelligence tools from the Unreal Engine to control enemy behavior and create a dynamic and challenging gameplay experience.

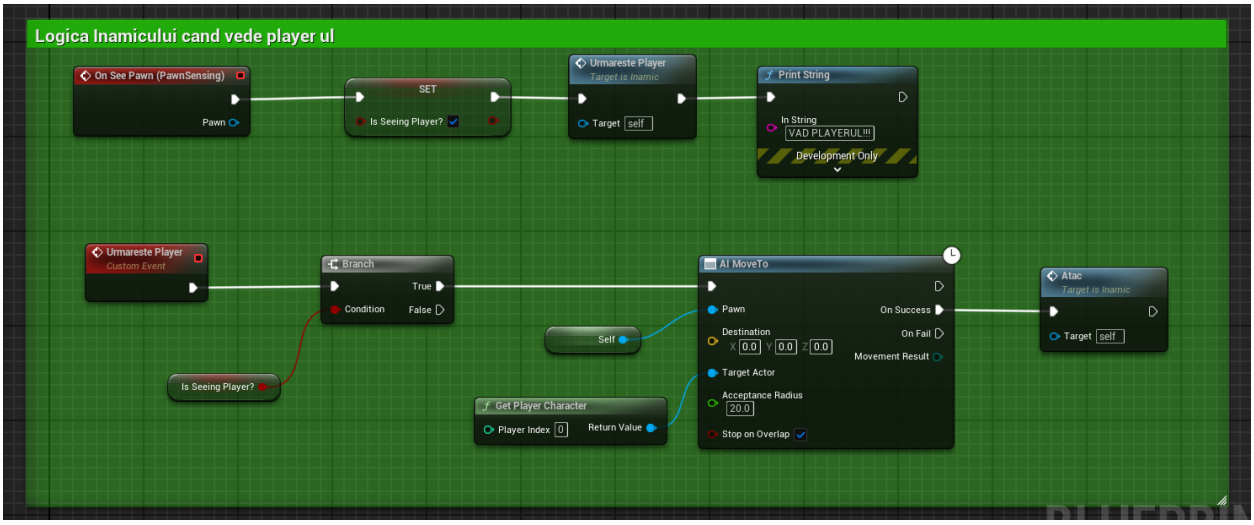
Every enemy in the game uses a**AIController**, responsible for managing its decisions and movement. For efficient navigation on the map, a **NavMesh Bounds Volume**, which delimits the areas accessible for movement. The size of this NavMesh has been adjusted so that enemies do not accidentally fall outside the map, thus ensuring fluid gameplay and free of navigation errors.

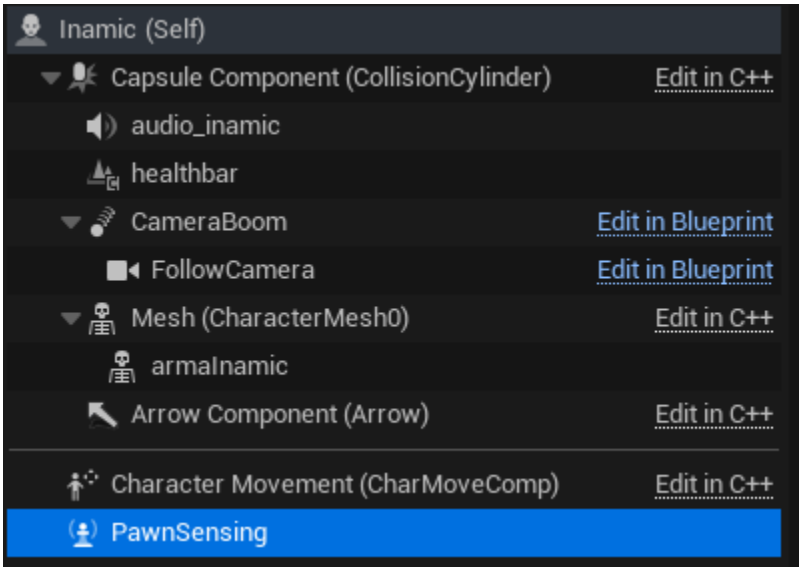
To move enemies, nodes such as**AI MoveTo**and **GetRandomReachablePointInRadius**, allowing for natural and varied movement across the map. Enemies do not follow predefined paths, but move randomly, making each encounter unique.

Player detection was achieved through the component**PawnSensing**, which allows the enemy to visually perceive the player's presence when they enter their visual range. This component allows for the creation of a realistic and reactive tracking system, increasing the difficulty and immersion in the game.

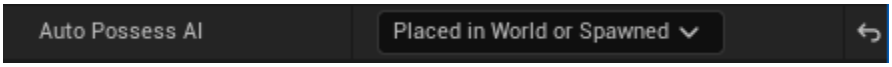
By using these artificial intelligence tools, EcoSurvivor manages to create a dynamic and unpredictable environment, offering players a constant challenge with every game session.

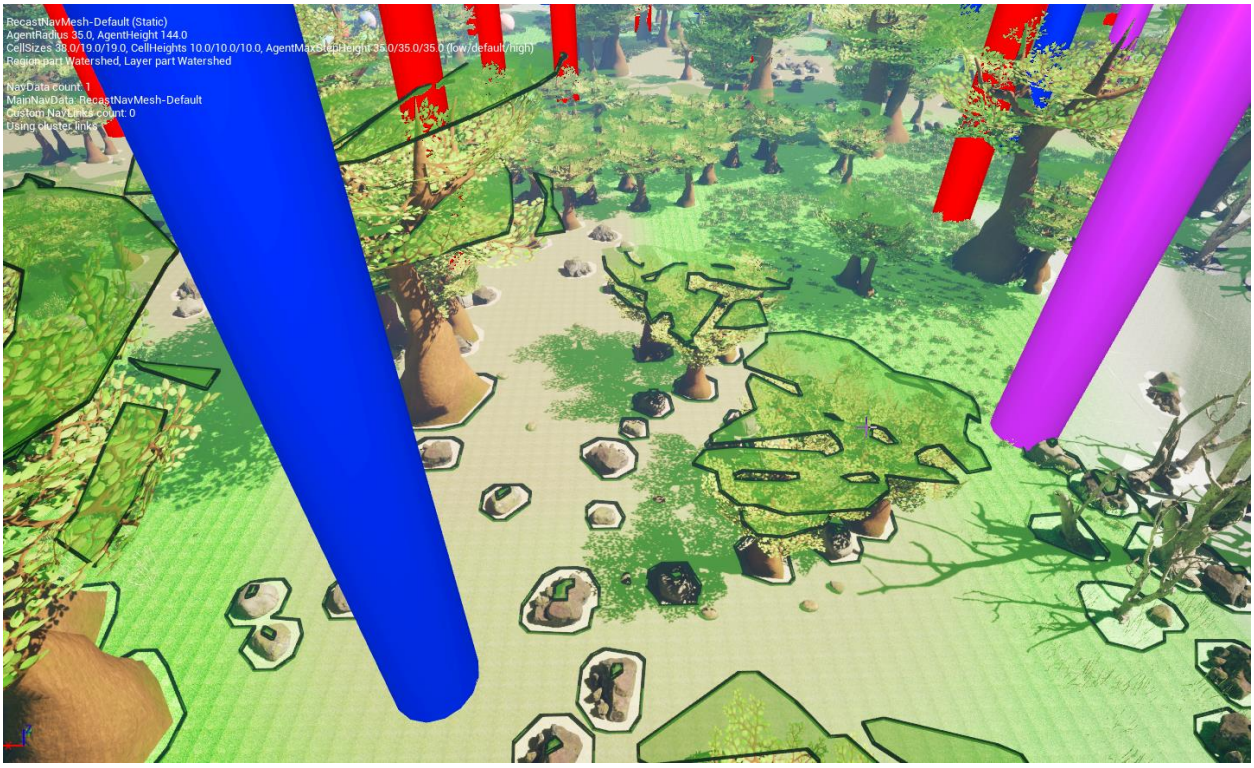



Logica Inamicului cand vede player ul
On See Pawn (PawnSensing)
Pawn
SET
Is Seeing Player?
Urmareste Player
Target is Inamic
Target self
Print String
In String
VAD PLAYERUL!!!
Development Only
Urmareste Player
Custom Event
Branch
True
Condition
False
Is Seeing Player?
Self
AI MoveTo
Pawn
Destination
X 0.0 Y 0.0 Z 0.0
Target Actor
Acceptance Radius
20.0
Stop on Overlap
On Success
On Fail
Movement Result
Get Player Character
Player Index 0
Return Value
Atac
Target is Inamic
Target self


Inamic (Self)
Capsule Component (CollisionCylinder)
Edit in C++
audio_inamic
healthbar
CameraBoom
Edit in Blueprint
FollowCamera
Edit in Blueprint
Mesh (CharacterMesh0)
Edit in C++
armaInamic
Arrow Component (Arrow)
Edit in C++
Character Movement (CharMoveComp)
Edit in C++
PawnSensing


AI Controller Class
AIController


Auto Possess AI
Placed in World or Spawned



List of completed tasks

Stage	Period bonus	Period normal score	Since I started penalty weekly
Stage 0 (recommended score 0.5) - topic choice and description	-	has no bonus; sent by 14.10.2024 because if not you have the theme of You don't have a project. how to work in stages next	

Tasks stage 0 (0.5))

- (0.~~02) Choosing a theme (finding a title for the game)~~
  Brief description of the topic (google docs, 1/2-1 page). It will contain the following information:
  - (0.~~13) a general description of the game (0.13 divided into) :~~
    - (0.~~05) Context (the setting, the story of the game; what it is about)~~
    - (0.~~03) What category it falls into (strategy, shooter, adventure, fighting, etc.). It can be a combination of categories.~~
    - (0.~~05) Game rules: what are the possible actions of the player. In what situations does the game end? How is it decided whether he has won or lost, how is the eventual score calculated.~~
  - (0.~~1) Establishing user interaction. Can be given as a description in words or UML diagram (only for actions) main ones, not the whole game). This should include usage scenarios. What is the first interface the user sees. What actions are available when they enter the application, etc.~~

- (0.02) Identification of the user category (age, personality, interests, social background).Example: the game is dedicated vegetarians of all ages, with an average to high level of general knowledge, passionate about cooking, strategy and ecology
- (0.03) Establishing keywords/phrases (a list of at least 5-6 keywords that describe the game). Reason: some game publishing platforms require such keywords, tags, etc.
- (0.2) Search for similar games (4-5 games) as a theme to observe the pros and cons (minimum 2 positive aspects and minimum 2 negative aspects for each game. Games can be evaluated based on a demo, watching streaming or free videos showing the gameplay, and even with the help of reviews. It is not necessary downloading/installing games. For each game you will write the name followed by comments.For example if you want to do a village simulation game you will list simulation games for some kind of community (village, city, tribe, etc.) The games found do not have to be identical in idea to your game but just have some similar features.

Score: 0.5

Stage 1 (recommended score 0.7, maximum ...) - simple scene - terrain, materials, foliage	11.11.2024 - 25.11.2024	26.11.2024 - the last day before on vacation winter	last 2 wEEKLY FROM semester

Stage 1 (Static elements) Recommended score 0.7. Attention, points are only taken on what is implemented by the student, not on elements imported from various packages!

Land requirements

- (0.05) There must be a terrain in the scene. It is not mandatory to move on the terrain, it can serve as scenery around the game platform.
- (0.15) The terrain must have a varied relief (there must be multiple low and high areas). The terrain will have a material assigned that includes multiple (minimum 3) textures (e.g. grass texture, sand texture, rock texture, etc.). The associated textures must be painted on the terrain so that they are consistent with the shape of the terrain (e.g. a deep hole will have a rock texture and not grass/flowers)
- (0.05) There must be at least one ramp on the terrain (Sculpt > ramp menu)
- (0.05) There must be two symmetrical areas on the terrain (for example two mountains). (see menu Sculpt > mirror) Foliage

requirements

- (0.1 0.5) The Folliage mode will be used to add scenery to the scene. The student will explain how they used the folliage mode. The score is given based on the complexity of the settings, the decor, the different types of instances used, and the student's ability to modify the scene during the presentation. (0.2)

Material requirements

- (0.05) Object with transparent material
- (0.05) Object with metallic luster that reflects the environment
- (0.1) Object with shiny material (which reflects the environment) in certain areas and matte in others according to a certain pattern (the solution will be done through blueprints)
- (0.05) Existence of an object with emissive color.
- (0.05) Object that has a 2-color gradient material (the gradient will be created through blueprints without using an external texture)
- (0.05) Object that has gradient transparency
- (0.15) Object that has a 3-color gradient material (the gradient will be created through blueprints without using an external texture)
- (0.1) Object that has a material that is transparent in parts and opaque in others. The transparency portions are decided by a pattern (e.g. another texture)
- (0.1) combining the colors of two textures according to a pattern given by a third texture
- (0.2) simulating a glitter color (randomly arranged glitter dots) using a noise node and without using an external texture (i.e. an image)
- (0.15) A complex material with a pattern created using at least 5 mathematical functions.
- (0.2) An animated material using the Time node, and at least 2 nodes with mathematical functions.
- (0.05) Using a normal map to create an object that gives the impression of having asperities even though it hasn't modified its vertices

Score: 1.65

Pawn/Character; movement through the scene, player progress, recommended: 0.75, maximum: 2.35)

A pawn and/or character will be created for user movement. **Strict requirements for pawn**

- (0.1) Making a pawn by extending the Pawn or DefaultPawn class

Strict character requirements

- (0.1) Character realization by extending the Character class
- (0.2 0.5) Create a character animation(s) to be used in the game. Points are awarded based on how complex the animations are, the context in which they are used
- (0.1) The class for the pawn/character in the scene is made in C++
- (0.1 0.3) The character must have actions (Action Mappings) defined in inputs in Project Settings and implemented in the blueprint (examples of actions: jump, crouch, fly, etc.)
- (0.1) Association of animations for the above actions (minimum one association)

Common pawn/character requirements:

- (0.05) The pawn/character will have a (recording) camera added to the components to follow the pawn in first person or third person style.
- (0.15) Ability to switch from first person to third person tracking by pressing a key.
- (0-0.4)**Creating variables for pawn/character or other actors**, which reflects the player's state, certain properties (Each type **different**of data from those listed 0.05). The score is given**only if the stored information is relevant to saving the game**: (0.25)
- Integer (integer or Integer64)
- Boolean
- Rational (Float)
- String (String or Text)
- Vector (e.g. for storing a color, location, etc.)
- data array (Array) of any type
- set of any type
- map of any type
- (0.1) The pawn/character must have Axis Mappings defined in the inputs in Project Settings.
- (0.1) It is added to the score if it can be translated on at least 2 axes defined as follows
- (0.1) It is added to the score if it can rotate about at least one axis
- (0.05) It is added to the score if at least one*mapping*it is made for mouse
- (0.1) The pawn/character can change (increase/decrease) its movement speed
- (0.1) The collision of the pawn/character with other objects will be handled using a*box*collision. The pawn/character may be able to pass through certain objects but not others (in at least one of these situations, one or more attributes of the pawn/character will change: for example, its health decreases if it touches an enemy)
- (0.05-0.1) A scoring system. Based on the achievements in the game, a number will be calculated to show how well the player did.
- (0.35) Highscore system. For games with a purpose, after the game is finished, the score will be saved in a file, with the scores of all users, ordered from best to worst (possibly the number of saved scores can be a maximum of N, and any performance below the first N will not be saved). The user will have the option to view the scores.

Damage system (recommended 0.25, maximum 0.25)

- (0.25) The default system will be implemented*damage*from Unreal either on the pawn/character or on the actors with whom the player interacts. The ApplyDamage method will be used following an event in the game. With the help of an AnyDamage event the actor on which the damage is applied will have some parameters affected. We will implement a case for a low value damage (the object can change its color, shrink, etc.) and another for a high-value damage (for example, the object can disappear or change its color differently than the low damage, or we can provide a message written on the screen).

Score: 1.85 / 2.35

Stage 3 - Trajectories. Projectiles (recommended: 0.5 maximum 1.75)			-

Trajectories (recommended: 0.5, maximum 1.75)

- (0.2) Static trajectories. There will be trajectories in the scene defined by spline curves created statically using the editor. On the trajectory there will be move actors
- (0.1) Speed that can be accelerated/decelerated along the entire trajectory
- (0.25) Accelerated/decelerated speed only on certain portions of the trajectory (predetermined or calculated by the program)
- (0.05) Stopping the path movement
- (0.05) Restarting the path from the point where the object started
- (0.05) Restart the walk on the trajectory from the initial point
- (0.05) Infinite walk on a trajectory.
- (0.1) Traveling the path a finite number of times (N) after which a certain action in the game occurs
- (0.3 + bonus 0.1) Generate a spline curve dynamically, programmatically in blueprint. If done in C++, you will receive 0.1 bonus points
- (0.2) Dynamically changing the trajectory following an event.
- (0.1) Existence of multiple objects on the same trajectory
- (0.1) Dynamically adding (following an event or a certain game state) additional objects to the trajectory.
- (0.1) Changing the direction of travel on the trajectory, by program.

Projectiles (recommended: 0, maximum 1.1)

- It is scored separately in addition to other scoring categories achieved by the projectile implementation (that's why some scores seem low, because we considered that they add up to points on trajectory and collision events)
- (0.2) Implementation of a special actor with the role of a projectile. It will be launched on a trajectory following an event. The projectile will start from an actor (it can also be a pawn/character) with the aim of reaching certain coordinates. The projectile disappears when it reaches a target (another actor)
- (0.2-0.4) The projectile can track a moving target (adjust its trajectory according to the target's coordinates). The score depends on the type of tracking and the naturalness of the trajectory
- (0.1-0.5) Application of physics to the projectile: projectile affected by the force of gravity, wind effects, precipitation, different friction force in different environments

Score: 0

Stage 4 - advanced scene - game architecture, level. Lights. (recommended 0.5; maximum 5.8p)			-

Scene (map) architecture (recommended: 0.2, maximum: 3.4p)

- (0.1-0.5) is given for **the complexity of the stage construction** (number of elements, placement method, constructions created by placing elementary shapes to obtain more complex shapes). Using the Foliage mode to create certain areas.
- (0.1-0.5) It is given for **programmatically generating actors with specific locations, rotations, sizes** in order to create constructions complex (example: a chessboard made of cubes, a maze, a house made of wall and roof objects that were placed via blueprint to get the look of a house). Generation of actors in the scene will be done in the blueprint with methods like Spawn Actor from Class. At least one characteristic of the actors will be calculated via blueprint (e.g. location, rotation)

Game levels (maps):

- (0.2) The game is multilevel with different maps. You move from one level to another following in-game achievements.
- (0.1-0.5 per level; max 2p for 4 levels) up to a maximum of 0.5 is given for each additional level, up to a maximum of 4 levels (the first level is scored in other scoring categories) depending on the complexity of its architecture (in terms of terrain, skybox (or skysphere), lights, objects, statically placed on the map (using the editor) or dynamically (by program) skybox/skysphere, atmospheric elements, etc.).
- (0.2) Using sublevels for map optimization

Score: 0.2 / 1

Lights and shadows (recommended: 0.2, maximum:1.8p)

- (0.05-0.1) Relevant use of at least one directional light source (Directional Light). Modification (static, manual) of its properties.

● (0.05-0.1) Relevant use of at least one point light source (Point Light). (Static, manual) modification of its properties.

● (0.05-0.1) Relevant use of at least one spot light source (Spot Light). (Static, manual) modification of its properties.

● (0.05-0.1) Relevant use of at least one rectangular light source (Rect Light). (Static, manual) modification of its properties.

● (0.05-0.1) Relevant use of at least one atmospheric light source (Sky Light). (Static, manual) modification of its properties.

● (0.05-0.1) Associating lights (as components) to actors (example: creating a flashlight, associating a Spot Light to the flashlight mesh)

● (0.1-0.5) Modifying the characteristics of lights (such as color/intensity, being on/off) based on events/player state/time in the game. Points are awarded based on the number of lights affected, the number of different types of modifications, and their complexity. Some examples (to give you an idea,but you can come up with something new) :

● if the game simulates day/night sequence, SkyLight may vary

● Or we have some lights that are on for 5 seconds and off for 2 seconds, then on again and so on)

● light that turns on when we enter a room

● (0.1-0.5) Animating the lights by changing the direction of the position, the attenuation distance, gradually and continuously. It is scored in depending on the number of lights affected, the number of different types of animations and their complexity. Some examples (to give you an idea,but you can come up with something new):

● A point light source that moves along a path

● Simulation of a star whose brightness varies periodically over time, gradually going from an intense light to a weak one, and so on.

● A rotating spot source, like an automatic spotlight

● Directional light source that gradually changes direction

● (0.05) Light source that does not cast shadows

● (0.05) Object (actor) that does not cast shadows even though other objects lit by the same source do.

● (0.1) Object not affected by light (Unlit material)

Score: 0.4 / 1.4

Random (recommended:0.1 maximum:
0.6) Using random numbers in:

● (0.05) generating a random color applied to an object (actor, widget, etc.) in the game

● (0.05) random coordinates, rotations and/or dimensions for one or more objects or pawns/characters

● (0.1) random string - for example for a password or part of the default username, or for saving the game

● (0.1) Randomly mixing the elements of a vector then used in the game

● (0.2-0.3) Probabilistically determined behaviors (0.2 is given for 2 complementary probabilities and 0.3 for more). Example: with a probability of 20% to generate elements of color c1, with a probability of 30% color c2 and the rest of color c3. Any element can be chosen that depends on the probability (color, location, shape, type of object, action performed, etc.) (0.2)

Score: 0.35

Score: 0.95 / 2.75

Stage 5 Events. Collisions. Time Manipulation (recommended 0.6 maximum: 2.5)			-

Mouse (recommended 0.1 maximum: 0.3)

● (0.1) Relevant use of a click event within the game

● (0.1) Relevant use of a begin cursor over event within the game

● (0.1) Relevant use of an end cursor over event within the game

Score: 0.1

Keyboard (recommended 0.1 maximum: 0.3)

● (0.1) Relevant use of a keydown event in the game

● (0.1) Relevant use of a keyup event in the game

- (0.1) Handling a key combination (between a special key - shift, ctrl, alt - and a displayable one, for example Shift+q, ctrl+w, etc.)

Score: 0.3

Collisions (recommended 0.2 maximum: 0.6)

- (0.1) Relevant use of an overlap event within the game
- (0.1) Relevant use of a hit event within the game
- (0.1-0.4) Updating pawn/character and/or actor data upon collision (hit/overlap) Points are awarded based on complexity collision handling, For example, if an actor (it could even be the pawn) is in collision with different actors (or different types of actors) different actions may occur (for example, when colliding with an energy bar, the bar disappears and the pawn gains health, but when colliding with an enemy, the enemy only changes color and the pawn loses health).

Score: 0.2 / 0.5

Time manipulation (recommended 0.2 maximum: 1.3)

- (0.1) Performing an action at a time interval t after an event has happened (For example, 2 seconds after the game starts, something happens), for example with a node of type "Set Timer by Function Name"
- (0.1) Repeating a function call at equal time intervals, with a "Set Timer by Function Name" node
- (0.1) following an event or a state reached by the game, a function called repetitively (at equal time intervals) using "Set Timer by Function Name", will be put on hold with "Pause Timer by Function Name". The repetitive call will be resumed following another event, with "Unpause Timer by Function Name"
- (0.05) following an event or a state reached by the game, a function called repetitively (at equal time intervals) using "Set Timer by Function Name", will be canceled (repeated calls will be permanently stopped) with "Clear Timer by Function Name".
- (0.2) Displaying the date (for example, in a widget) using the now node and breaking the DateTime structure into components. The date will be displayed in day/ month/year format (and if a number is less than 10, it will be preceded by the digit 0)
- (0.2) Displaying on the screen, during the game, the time that has elapsed since the start of the game or the start of the session, or since a certain event.
- (0.3) For time information (how many seconds remain until an event or how many seconds have passed since a time t, the time instead of being displayed as an integer number of seconds will be displayed in the format hh:mm:ss (h - hour, m - minutes, s - seconds). If any number in the 3 categories is below 10, it will be displayed preceded by a 0).
- (0.25) Displaying the last access time or the time elapsed since the last access in a widget.

Score: 0.95

Score: 1.55 / 1.85

Step 6 - Menu. Adding sounds (recommended: 2.2p maximum: 6.35)			-

Game menu (recommended: 2p, maximum: 5.35)

- (0.2p) The game's main menu will be displayed upon entering the game. The game is not started (i.e. paused) until the menu is exited.
- (0.05) Using an image in a panel
- (0.05) Using HorizontalBox and/or VerticalBox panels
- (0.2p) Switching between menu screens using WidgetSwitcher
- (0.2-0.4p) Creating a custom class for buttons. (Points are awarded based on how complex it is.

The main menu may contain the following buttons:

- (0.2p) The button to start a new game. The button will have a suggestive text, for example "Start". Upon entering the application, the game is paused, and remains so until the user activates it
- (0.3p) The button to continue the last game started. When clicking on this button, it starts the game displaying the exact state in which the user left it before the last close (or last save)
- (0.2p) General settings screen (for player profile or new game features. The settings screen will be accessed via a button on the main menu. The settings screen will contain various inputs and a button to send data. When you click on the button, the settings will be saved in the pawn/character properties.

Inputsused in the settings screen (or other screens that request information from the user can be of the following types (note, points are scored for each distinct type, not the input itself):

- (0.05) EditableText
- (0.05) TextBox
- (0.1) Slider. Parameters such as: minimum and maximum values and step will be set.
- (0.1) SpinBox. Parameters such as: minimum and maximum values, number of decimal digits, step (delta), growth exponent (increase step for larger values) will be set.
- (0.05) Checkbox
- (0.1) ComboBox with at least 2 options
- (0.1) RadialSlider Parameters such as: step, default value, etc. will be set.
- (0.1) An additional 0.1 is given for the ComboBox if the options are added dynamically.
- 0.1 is added separately for each input type that is included in a custom widget in order to add new functionalities to the widget (Example: a borderbox that changes its background color with each keystroke)

- (0 - 0.3) Appearance of the settings screen. Points are awarded based on:
  - Most important: Aligning elements (e.g. with a grid)
  - The fact that each input has an associated label (text next to it that says the role) and/or tool/type,
  - Changing default colors
  - Readable text (chosen colors with good contrast; text is not too transparent or overlapped with an image)
  - Using a pleasant background.
- (0.1) Dynamically (programmatically) adding elements to the widget, using methods like "Add Child to [container]"
- (0.1) Dynamically (programmatically) deleting elements in the widget, using "Remove Child" methods
- (0.4p) The button to load an old game. When clicking on this button, a screen will open with the user's previous saves from which he can choose which game he wants. The saves can be listed through buttons or through a combobox. The screen will be dynamically generated depending on the files in the saved games folder. The games that correspond to the current player will be identified by username.

- Game information display button:
  - (0.1) will take you to a screen with some game text explaining the story/background. The screen has a button to return to the main menu.
  - (0.4) Special stylization of the game text screen. Points are given based on how complex and beautiful the stylization is:
    - using a scrollPane
    - using different colors within the text
    - using different styles: bold/italic
    - stylizing text in the form of sections with headings
    - using lists
    - using images within the text
- (0.1p) Exit button from the application (clicking on it closes the game)
- (0.1) Using a widget, a menu will be created that will be displayed during the game and will have the buttons (additionally dotted as follows):
- (0.1) Pause - when you click on it, the game pauses, and when you click on it again, it restarts. The button text should differ depending on the type of pause, for example, it should say "Restart"
- (0.1) An exit button.
- (0.2) A button/key shortcut that pauses the game and displays the main menu. In this situation, the menu should have an additional button with the text "Continue".
- (0.1 0.3) Displaying information related to the player's (or other actors') status during the game. Points are awarded based on how complex the display is.
- (0.1) Using a progress bar to display player information
- (0.2) Option to hide and redisplay the display during gameplay, for example, when pressing a key.
- (0.2) Simulating radio buttons using custom buttons
- (0.3) Simulating radio buttons using custom checkboxes
- (0.2 0.5) One or more informational screens that appear following an event or state the game reaches. For example, a screen informing the player that they have entered a new level or that they have "died". They are scored based on their number and the complexity of the display.

- (0.1) Restart button in an information screen, in case the player died or the level ended
- (0.2-0.4) Loading screen - scored based on complexity

Score: 2.2 / 2.7

Sounds (recommended: 0.2 maximum:0.5)

- (0~~.1) Adding a sound to the game Additional~~ points are awarded:
- (0~~.1) The sound occurred within a widget following an event (e.g., button click)~~
- (0.~~1) The sound occurred as a result of a collision~~
- (0.1 ~~0.2) The sound depends on the player's actions and the environment: moving through sand generates a different sound than moving through puddles)~~

Score: 0.4 / 0.5

Score: 2.6 / 3.2

Step 7 - save the game (recommended: 0.5, maximum:1.5)			-

Saving and reloading the game (recommended: 0.5, maximum:1.5)

- (0~~.1) Creating a class derived from SaveGame~~
- (0~~.2) Option to save the game while playing to a file, without going through the main menu. For example, by pressing a key combination or clicking on special elements in the game.~~
- (0.2) Additional points are awarded if, when saving the game, the user is asked via a widget whether they want to save over the current file corresponding to the current game session or want a new file, in which case they can opt for part of the name (for example, the save is in the form [username][timestamp][username]
- (0~~.4) Saving relevant information for restarting the game from the left point and displaying settings already made by the user. (Each type~~**~~different~~**~~of data from those listed 0.05). The score is given~~**~~only if the stored information is relevant to saving the game~~**~~:~~ (0.15)
- In~~teger (integer or Integer64)~~
- Boolean
- R~~ational (Float)~~
- S~~tring (String or Text)~~
- Vector (e.g. for storing a color, location, etc.)
- data array (Array) of any type
- set of any type
- map of any type
- (0~~.2) Creating a function in Blueprint to read the save file and set some data in the game.~~
- (0.2) AutoSave option - the game will be saved at every time interval t.
- (0.2) Saving at checkpoints. If the user manages to achieve something special like reaching a location or finding an artifact or defeating an enemy, the game is automatically saved and when re-entering the game it will start from the previous checkpoint.

Score: 0.65

Stage 8 - C++. Design patterns. Object-oriented programming.			

Stage 9. Visual effects: particle systems, post-processing effects.			

Stage 10 - documentation (recommended: 0.5, maximum: 1.1p)			-

Documentation (recommended: 0.5, maximum: 1p)

Note: write briefly and to the point, the number of pages does not matter, as long as the information requested below appears. A one-page documentation can be worth a maximum and a 10-page one can be worth 0.1-0.2.... You can also write "telegraphically" as long as you go through all the required sub-points.

It will include the following parts/chapters:

- (0.05) First page with name, surname, group, exam date, and game title. At the bottom of the page, the name of the subject. A table of contents to chapters (if there are more pages). The names of the chapters are written in bold, black. The pages (if there are more) will be numbered.
- (0.1)**Detailed game description**(not the original plan but just what you managed to implement. What the game does, what the story is (this part may also contain fragments from stage 0). May contain printscreens from the game for clarification
- (0.05)**Technical specifications**: the operating systems for which the game was developed, how much disk space it takes up, the approximate memory it needs, additional packages that should be installed, whether or not it requires a network connection, what kind of data it saves and approximately how large a save file can be, the version of Unreal Engine on which the game was developed.
- (0.05)**Thematic specifications**(necessary for the eventual publication of the game): the category/categories of the game, the motivation for choosing the theme the game, a list of descriptive tags, whether the game is single or multiplayer, the playing time (estimated in how many hours the game can be completed; for unlimited games it will be specified that the playing time can be any length), the languages in which the game is available.
- (0.1)**Available actions. Control.**What actions are available in the game and with what input peripheral devices can we perform them? (mouse, keyboard, joystick, etc.), with what combination of keys and buttons. What shortcuts are there. What options does the user have to define their own combinations of mouse/keyboard events for various actions
- (0.1)**Own classes.**Description of the classes created within the game: their role, important properties, what methods they implement, how they are integrated into the game. What*design patterns*have been implemented (and a brief description of each*design pattern*).
- (0.1-0.2)**Algorithms.**Description of the algorithms used (e.g., algorithms for generating a structure, special algorithms for calculating the score, artificial intelligence algorithms that control pawns/characters in the game. Enumeration of elements implemented in C++. Screenshots with blueprints or sequences from C++ code will be inserted in the explanations. Complexity analysis and The efficiency of the algorithms in terms of both time and memory. The score depends on the number of algorithms, complexity and analysis method.
- (0.05-0.1)**Artificial intelligence tools in Unreal.**Description of the artificial intelligence tools (from Unreal) used and how they are applied in the game.
- (0.1)**List of completed tasks.****Required for presentation.**Here will be the requirements copied and pasted from the scale, which the student has completed them. The list of requirements will be organized into categories and subcategories exactly like the present scale. Next to each task the score from the scale will be written, and the student will make an estimated sum of the points under each category but also at the end,**for verification**(The scale is large and complex and it's easy to overlook something in the presentation, with this measure we can check if we haven't missed anything).
- (0.05-0.1)**List of utilities used.**You will list all the utilities used to create assets (models, textures, sounds, etc.) for the game. You will list the assets and briefly explain how you made them.
- (0.05)**List of external packages used.**You will list all the packages added to the game, all the assets taken from other sources (images, and in what context you used them. You will also provide a link to each of the packages.
- (0.05-1)**Bibliography.**You will list the books, tutorials (written or video), forums where you took ideas, pieces of code/blueprints. If the source is online, you will also write the link. For each source, you will specify what exactly you took from there.

Score: 1.1

Other scoring categories

Algorithmic (recommended: 0.5, maximum:2)			-

Algorithmic (0.1-2p)

Up to 2 points are awarded per algorithm depending on their number and complexity.

An algorithm is scored based on:

- relevance in the game

- time and space efficiency
- the special data structures used (vectors, matrices, graphs, neural networks)
- the mathematical formulas involved
- interaction with the in-game environment, with the player's state
- the number of tasks it performs
- scalability
- Note: generally those who have puzzle games or strategy games will have the most here. The course game for example would get about 1-1.5 because of the maze and enemies. A simple shooter, with only health calculation, small upgrades, etc., gets about 0.5-0.7. A game where nothing much happens (the man also tried something so as not to say that he has never opened Unreal in his life) gets 0.1

Score: 0.1 / 2

Aspect (recommended: 0.5, maximum:1)			-

Appearance, relevance and accessibility

Points will be awarded taking into account the following:

- Placing objects in the scene in a logical and orderly manner (for example, several trees grouped on a terrain area to form a grove and not suspended in the air or on a character's head - unless the game's story requires them to be there. You're going to make all the trees flying in the game now, right?
- The terrain has a natural, logical, neat, pleasant shape. For example, it's not just a flat terrain with three holes placed just for the sake of it.
- Sky Sphere has a relevant texture (we don't have clouds and sun for a game that takes place in the cosmos)
- Using appropriate colors (that do not bother the eye) and beautiful textures. Textures can be taken from the internet as long as copyright is not violated and the source is mentioned in the documentation.
- Materials match the mesh (for example, an object that we know is metallic will have a metallic sheen).
- Models are suitable in shape and size.
- The stage architecture and color scheme match the game theme and the targeted user category.
- The lighting of the scene is appropriate (for example, if we are in a room where we are looking for something, let's have a light source to distinguish objects (unless the story requires it)
- The widgets displayed are easy to use. The texts do not overlap. The contrast between the text and background color is appropriate (easy to read).
- The animations are appropriate, with smooth transitions, there are not too many to make the scene tiring.

Score: 0.5 / 1

Code organization (recommended: 0.5, maximum:1)			-

-

Score: 0.5 / 1

Originality / creativity (recommended: 0, maximum: 0.5)			-

It can be given for:

- **very interesting game ideas**
- **special implementations**
- **special, interesting (student-created) resources**

Score: 0 / 0.5

Bonuses/optional

Modeling (e.g. Blender, Unreal in modeling mode) (recommended: 0, maximum: 0.9)			-

● (0.1-0.3 per model; maximum 0.9 for 3 models) The models modeled by the student in Blender (or another modeling tool) are scored according to their complexity. Up to**maximum 3**different models (So present the most complex models; if the differences between the models are small, it is considered a low degree of complexity). The student must show proof of the creation of the models (for example, the files created by the utility, printscreens with the progress of the work, the evolution of the model along the way), the explanation of the method of creation in the documentation (what tools he used).

Score: 0

Sounds (recommended:0, maximum:1)			

● (0.1-0.2 per file; maximum 1p for 5 files) The sounds created/recorded by the student themselves are scored according to their complexity. in any tool they wish (for example, Audacity). Up to**maximum 5**different sounds (So present the most complex files; if the differences between the sounds are small, it is considered a low degree of complexity). The student must show proof of the creation of the sound files (for example, files created by the utility, printscreens with the progress of the work, the evolution of the track along the way), the explanation of the method of creation in the documentation (what tools he used).

Score: 0

External packages (recommended:0, maximum:1.4)			

External packages (recommended: 0; maximum: 1.4)

● (0.~~1 per package; maximum 0.3 for 3 packages) Adding an external package and using at least one mesh/texture/material from it. Points are awarded up to**maximum 3**different packages (0.3)~~

● (0.05-0.1) Modifying the parameters (except for positioning for actors) of at least one element introduced from the external package. Points are awarded based on complexity.

● (0.~~1-0.2 per class; maximum 1p for 5 classes) Extending a class of any kind within the package and modifying the parameters, adding new parameters/methods. Points are awarded based on complexity up to**maximum 5 classes**different modified (0.1)~~

Score: 0.4

Final score: 12.35 / 18.95 [minimum / maximum]

List of utilities used

For the development of EcoSurvivor, several platforms and tools were used to create and integrate the necessary assets into the game.

The 3D models used in the game were largely taken from**Sketchfab**(link) and **Mixamo**(link). **Mixamo**It was used especially for animations, offering a wide range of predefined movements that were integrated directly into the Unreal Engine.

The textures and materials used in the game were downloaded via**Quixel Bridge**(link) and **Fab.com**(link These resources allowed the creation of a rich and realistic visual environment, optimized for performance.

For the sound effects, free resources from**Pixabay**(link). These sounds have been selected and integrated to provide an immersive and authentic audio experience.

By using these platforms, high-quality assets were ensured, saving time in the development process and optimizing the overall performance of the game.

List of external packages used

Several external packages were used in the development of EcoSurvivor to enhance the visual appearance and functionality of the game. These were strategically integrated to add environmental elements, visual effects, and essential mechanics.

- **Advanced Village Pack**(link) – this package was used for decoration, especially for trees and vegetation elements that contribute to the atmosphere of the island.
- **Third Person (Unreal Engine 5)**–was used as the basis for the character's movement mechanics. The character class in this pack was expanded to create the game's main character.
- **Starter Content (Unreal Engine 5)**–Various textures from this pack were used to create the game environment and improve the visual appearance.
- **Realistic Starter VFX Pack Vol 2**(link) – this package was used for blood visual effects, adding a touch of realism to combat interactions.

Bibliography

Sources of inspiration and tutorials used

- **Create realistic water effect**–YouTube Tutorial
- **Crowbar: animations, line tracing, combat**–YouTube Tutorial
- **Character: animations, walking, jumping**–YouTube Tutorial
- **Random number generation**–YouTube Tutorial
- **Implement countdown timer**–YouTube Tutorial
- **Use Set Timer by Function Name**–YouTube Tutorial
- **Create main menu**–YouTube Tutorial
- **Save system (save file)**–YouTube Tutorial
- **Spawn point generation**–YouTube Tutorial
- **Collide trigger detection system**–YouTube Tutorial

Packages used

- **Realistic Starter VFX Pack Vol 2**–Link

- **Advanced Village Pack**-Link
- **Starter Content (Unreal Engine 5)**
- **Third Person (Unreal Engine 5)**

Platforms and external sources of assets

- **Mixamo (character animations, 3D models)**–Link
- **Sketchfab (3D models)**–Link
- **Fab (resources and assets for games)**–Link
- **3D Models (flare gun 3D model)**–Link
- **Quixel Bridge (textures, materials, assets)**–Link

Sounds and audio effects

- **Pixabay (free sound effects)**–Link