Informatics Engineering Degree's Final Project:

Development of a stealth game with multi-agent artificial intelligence

Albert Espín Román

Director: Francesc Xavier Dantí Espinasa

Academic year 2017-2018 (2nd semester)

University of Barcelona

Presentation contents

1. Introduction
   - Objectives and motivation
   - Technology, methodology and planning
   - Introduction to the developed game
2. Development
   - Core gameplay structure and characters' abilities and interactions
   - Artificial intelligence: multi-agent system, perceptions, reasoning and decisions
3. Conclusions
4. Demonstration
   - Cinematic introduction
   - Gameplay
5. Questions and answers

Part 1 - Introduction

Objectives

The project's objective was to develop a stealth video game with artificial intelligence-powered enemies.

Player's features:

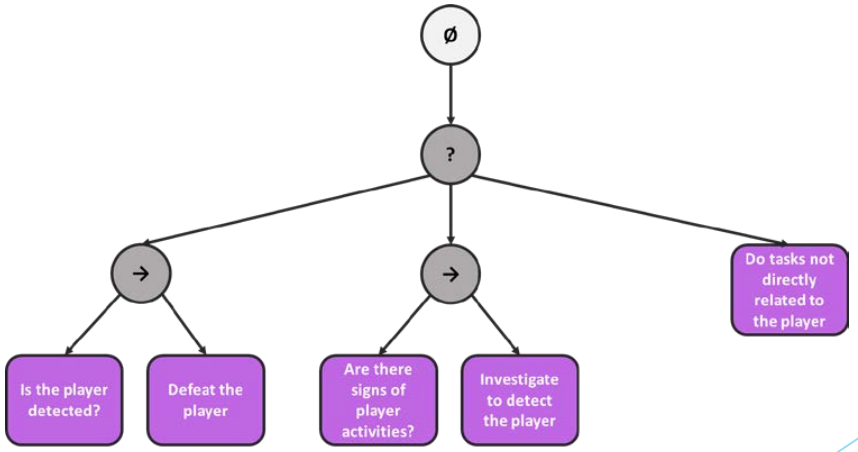
- Silent and careful to approach survive
- Use of objects to distract the enemies
- Ability to attack enemies from behind
- Need to infiltrate in hostile territory to recover valuable objects

Enemies' requirements:

- Ability to perceive the environment: vision and hearing
- Ability to reason and take decisions
- Shared goal of defeating the player
- Multi-agent system to cooperate and communicate information

Motivation

- This project would allow to:
  - Gain experience in the fields of game development and game-oriented artificial intelligence.
  - Learn how to design and implement behaviour trees to model reasoning and decision-making for non-player characters.



Technology and methodology

Unreal Engine 4 was chosen as the game's engine because:

- It is extensively used in the game industry.
- The developer had some previous experience with it.
- It has a built-in behaviour tree editor.

C++ was used to program the game in combination with Blueprints, Unreal's visual scripting system. Benefits:

- C++ gives a solid base for core classes and functionalities.
- Blueprints allow to prototype fast and customize higher-level details.

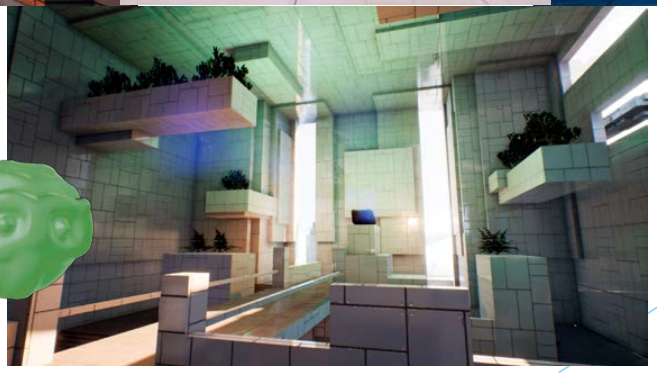
Planning

- An initial planning was set at the beginning of the project.
- The target features were divided in modules, which included different tasks.
- Artificial intelligence tasks were considered critical and the most complex, so they were a priority in every module.
- All 7 modules were completed in time and extra tasks were done.
- 9 videos were made during the development process to demonstrate the implemented features.

Introduction to the developed game: Stealth Temple

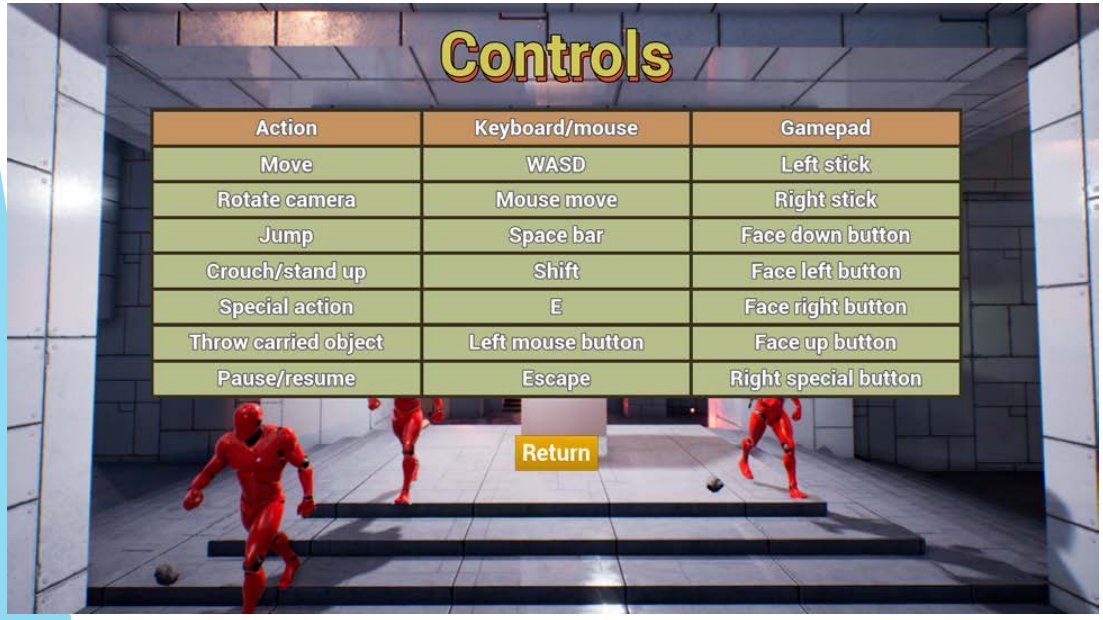
Player character

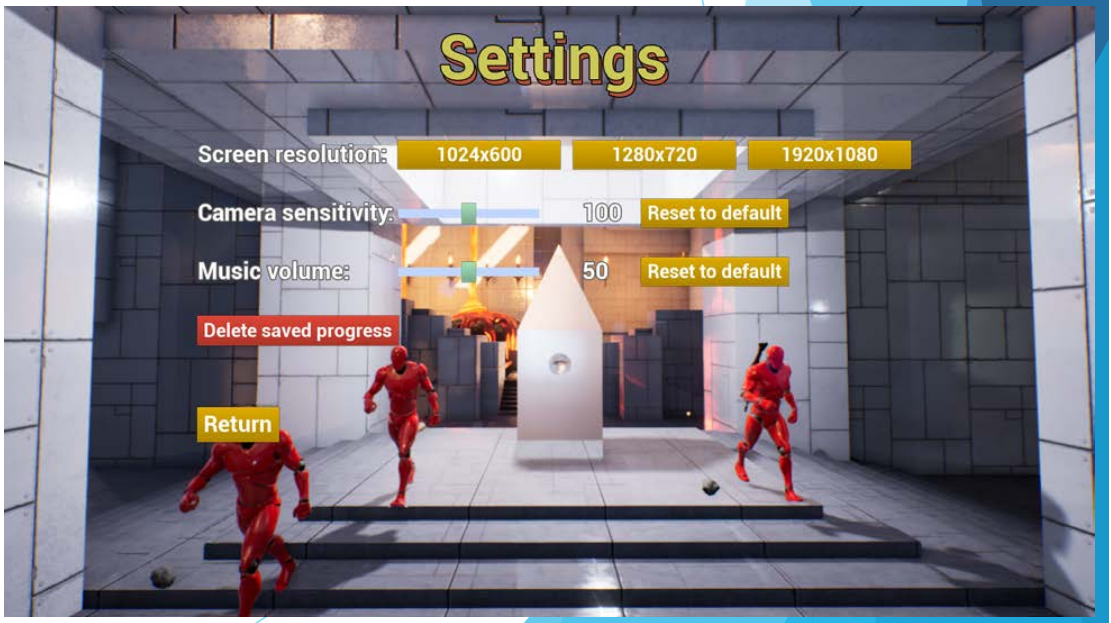
Enemy guards



Main Menu

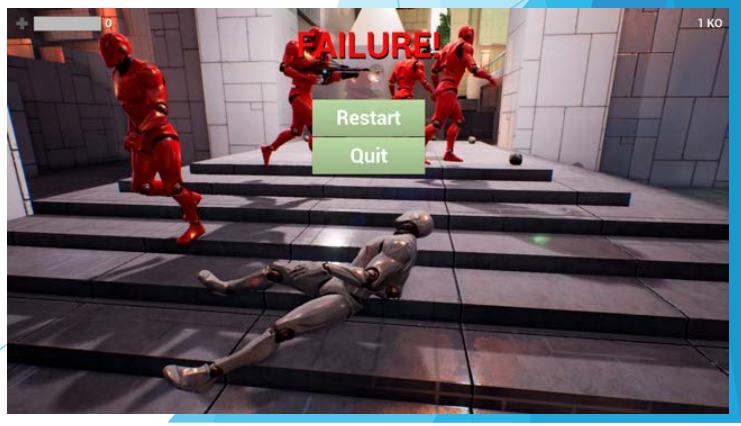

STEALTH TEMPLE
Play
Controls
Settings
About
Quit


Controls
Action
Keyboard/mouse
Gamepad
Move
WASD
Left stick
Rotate camera
Mouse move
Right stick
Jump
Space bar
Face down button
Crouch/stand up
Shift
Face left button
Special action
E
Face right button
Throw carried object
Left mouse button
Face up button
Pause/resume
Escape
Right special button
Return


Settings
Screen resolution:
1024x600
1280x720
1920x1080
Camera sensitivity:
100
Reset to default
Music volume:
50
Reset to default
Delete saved progress
Return

Gameplay basics

- 3rd person perspective
- Camera can be rotated
- Basic abilities: run, jump and crouch
- Seek the sacred orbs in the temple zones
- Bring the four orbs to the monolith to win
- Some enemies can shoot, others punch
- Enemies patrol the temple and cooperate to find and defeat the player



Throwing stones to distract enemies

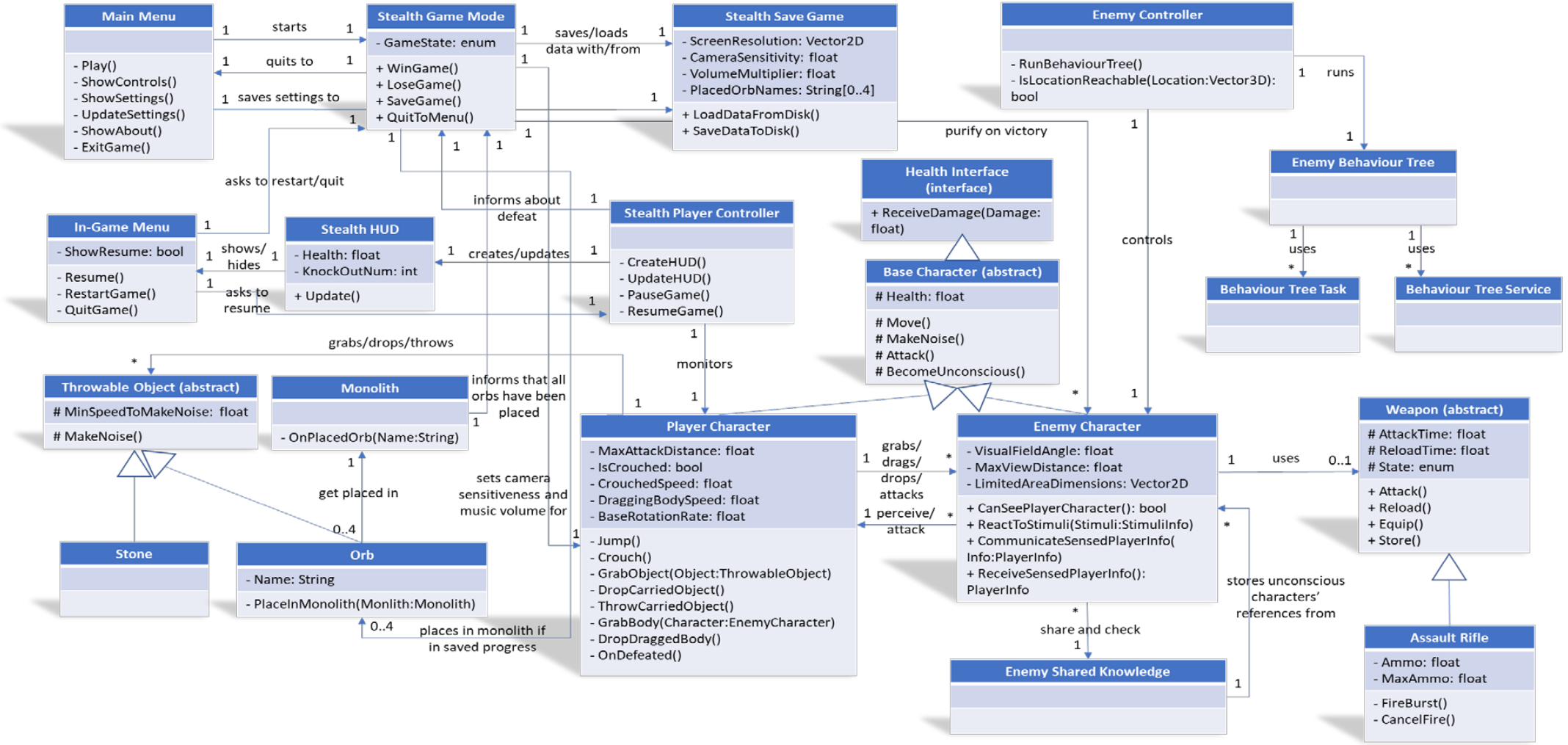
- Stones can be picked up from the floor and thrown to make noise and distract enemies.
- Enemies investigate the source area of a sound, so stones should be thrown from hidden locations.

Knocking enemies out and hiding their bodies

- Enemies can be knocked out from behind with a hand attack.
- This is a stealth move: it can only be used while undetected.
- Other enemies investigate the area where they see an unconscious body, so it is better to drag the body and drop it in a hidden place.

Part 2 - Development

Core gameplay framework: class diagram

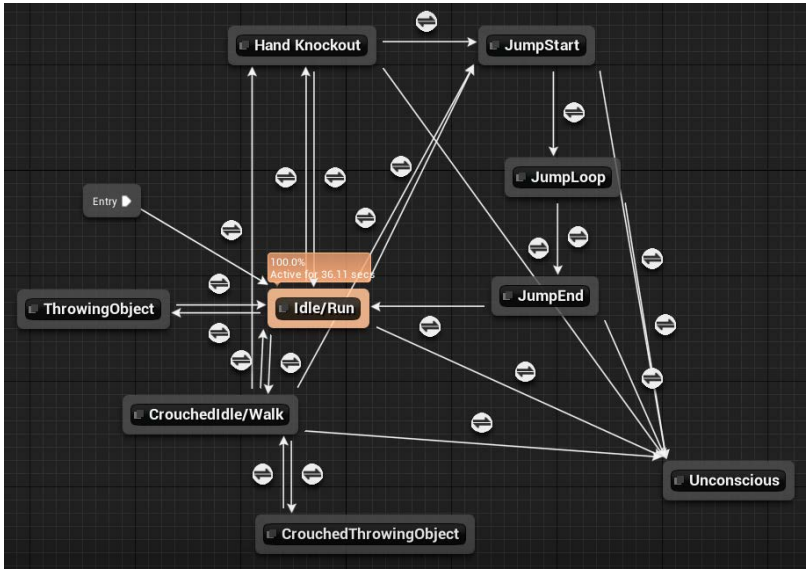


The characters' components

Characters are made up of components to build functionality:

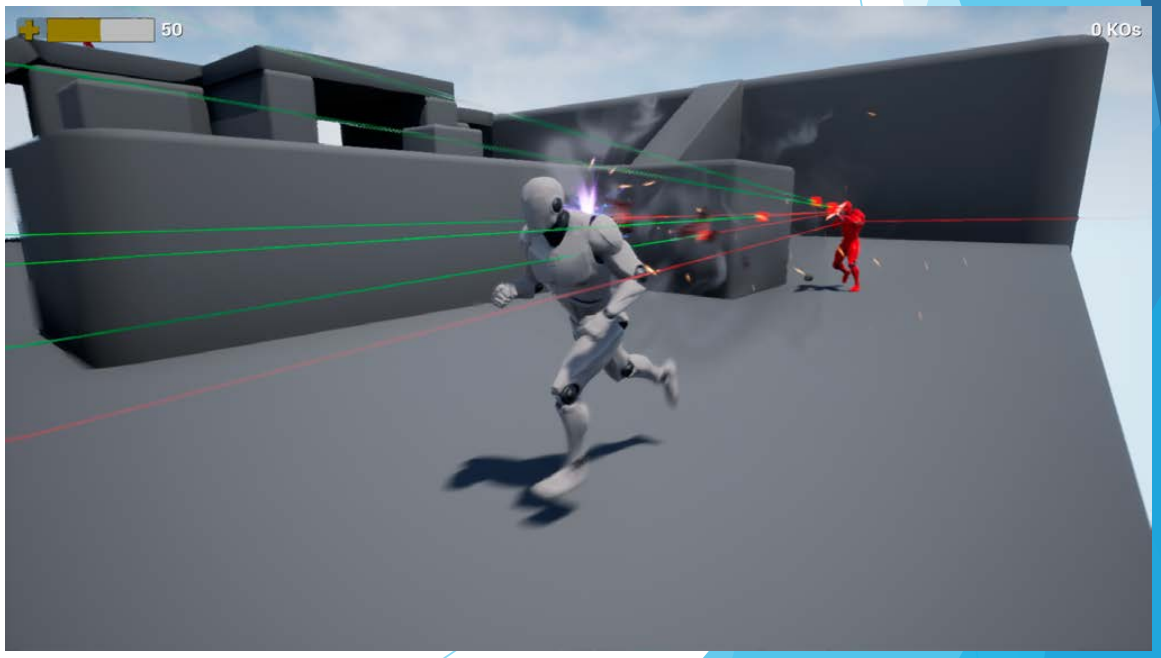
- Capsule component (root): implements the collision with the world.
  - Movement component: sets how the character can move, crouch and jump.
  - Camera component: sets how the camera works.
  - Mesh component: sets the physical appearance, linked to the animation logic and state machine.

State machine for the characters' animations



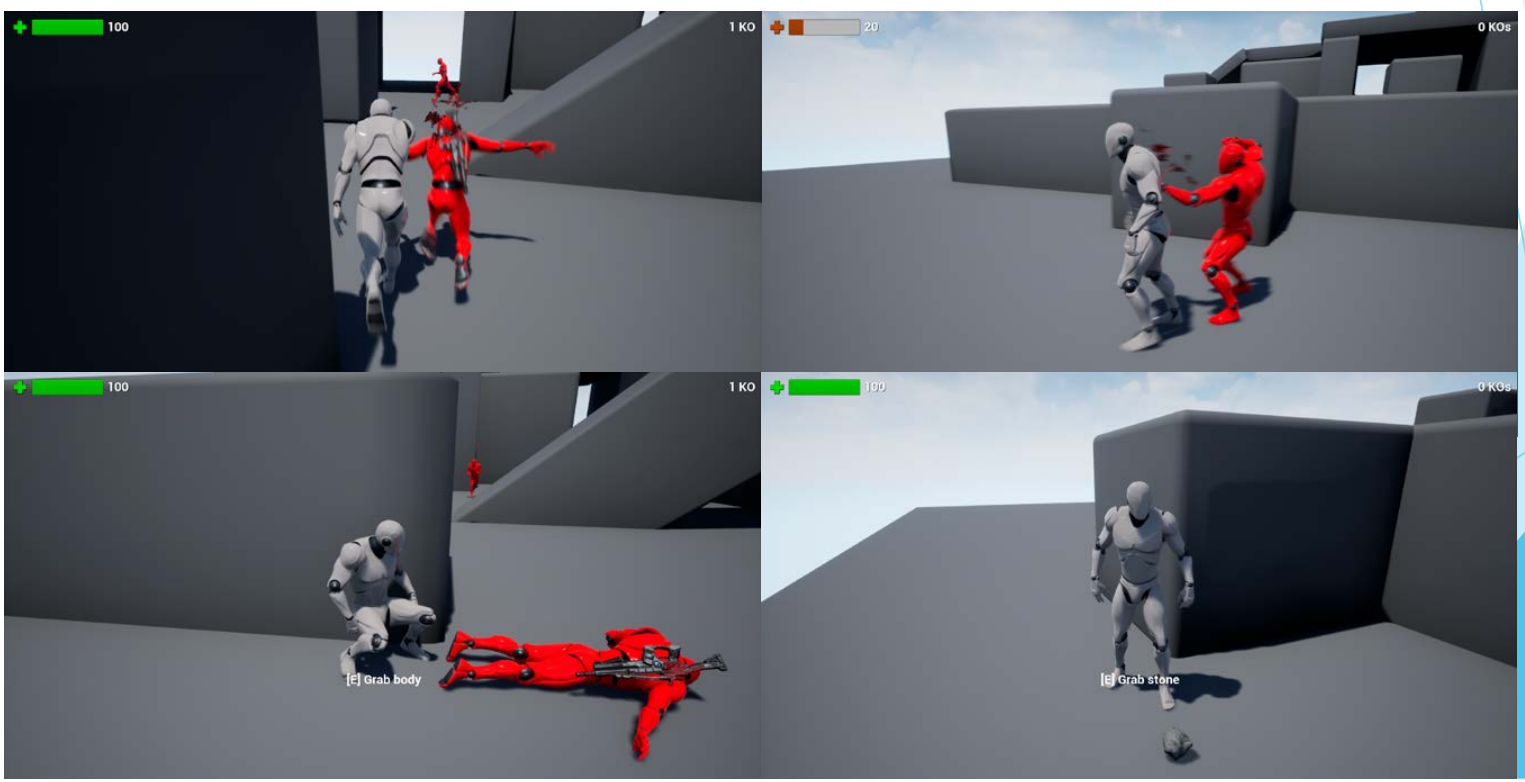
The characters' interactions with the world (I)

- Characters use ray traces and invisible volumes (spheres/ capsules) to test collision with the world. They are enabled when necessary.
- Events are triggered when the traces and volumes intersect with key game entities.
- In the case of the player character, messages are shown on the screen to notify the player of the availability of certain actions, e.g. grabbing a nearby stone.
- Ray traces are used to:
  - Estimate if an enemy's hypothetical shot in a specific direction would be blocked by an obstacle or not.
  - Determine if an actual shot hits the player character, to cause damage.



The characters' interactions with the world (II)

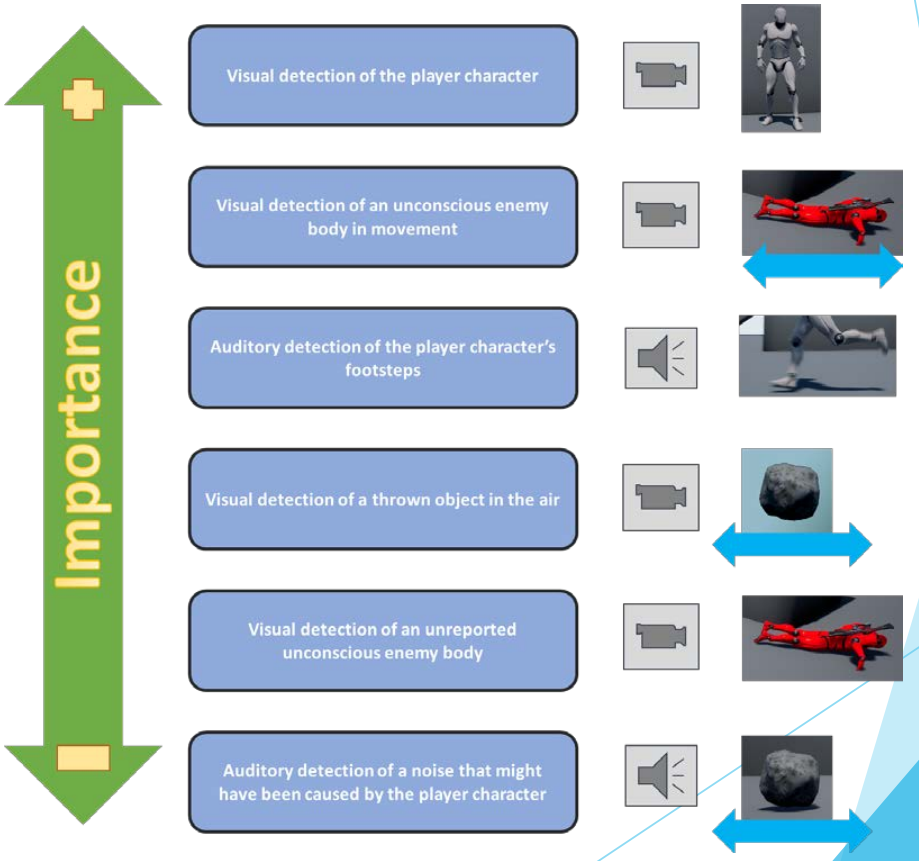
- Volumes are used to:
  - Detect when the player character hits an enemy with a knock-out move.
  - Detect when an enemy punches the player character to cause damage.
  - Detect near objects and bodies in the floor that the player can grab.
  - Detect the monolith so that the player can place an orb there.



The enemies' multi-agent system (I)

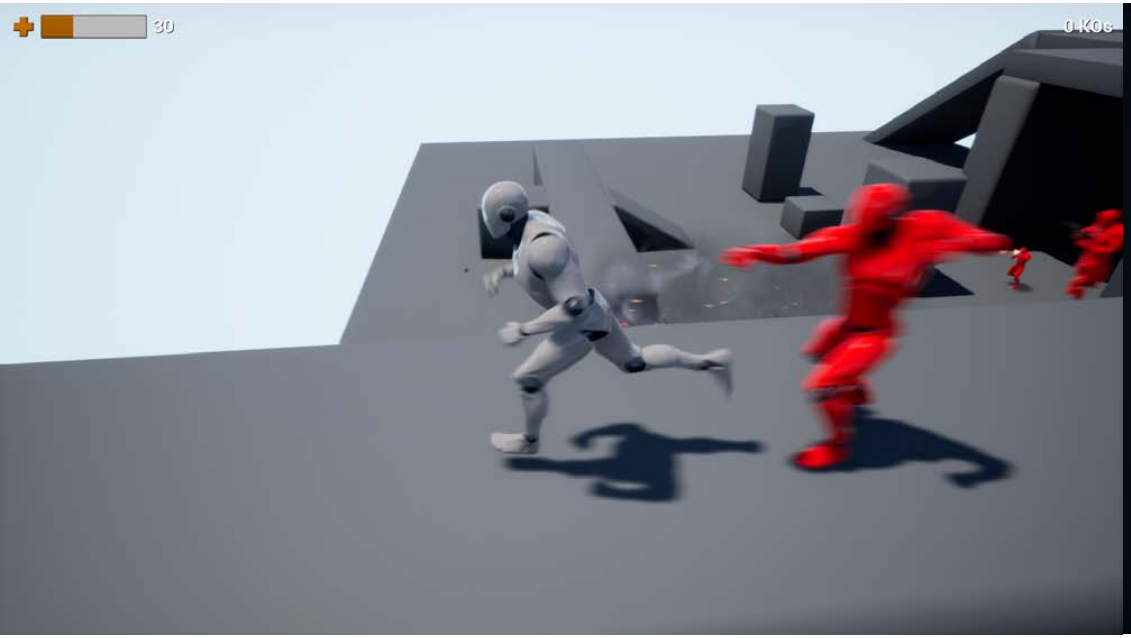
- The enemies form a multi-agent system to achieve the shared goal of defeating the player as fast as possible.
- Intelligence is decentralized: individual enemies take independent decisions, but they take into account what other agents say.
- Every time an enemy perceives stimuli that may be related to the player, the information is sent to other agents.
- Information is used to take decisions using a hierarchy: the more certainty a piece of data gives about the current player location, the more valuable it is.

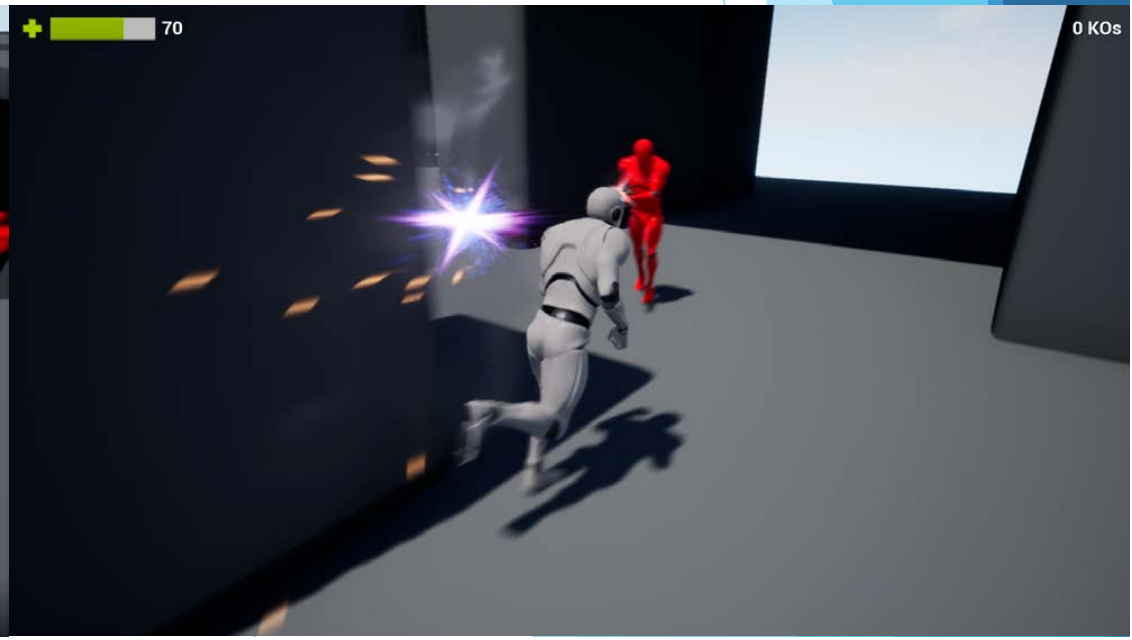
Types of information about the player that enemies share



The enemies' multi-agent system (II)

- Dynamic roles: some enemies focus on chasing the player character to track the player's movements, so others can focus on aiming and shooting at the head with high precision from distant positions.
- Static roles: some enemies patrol the whole environment (chasers) while others protect limited sensible areas and are harder to distract (guards).

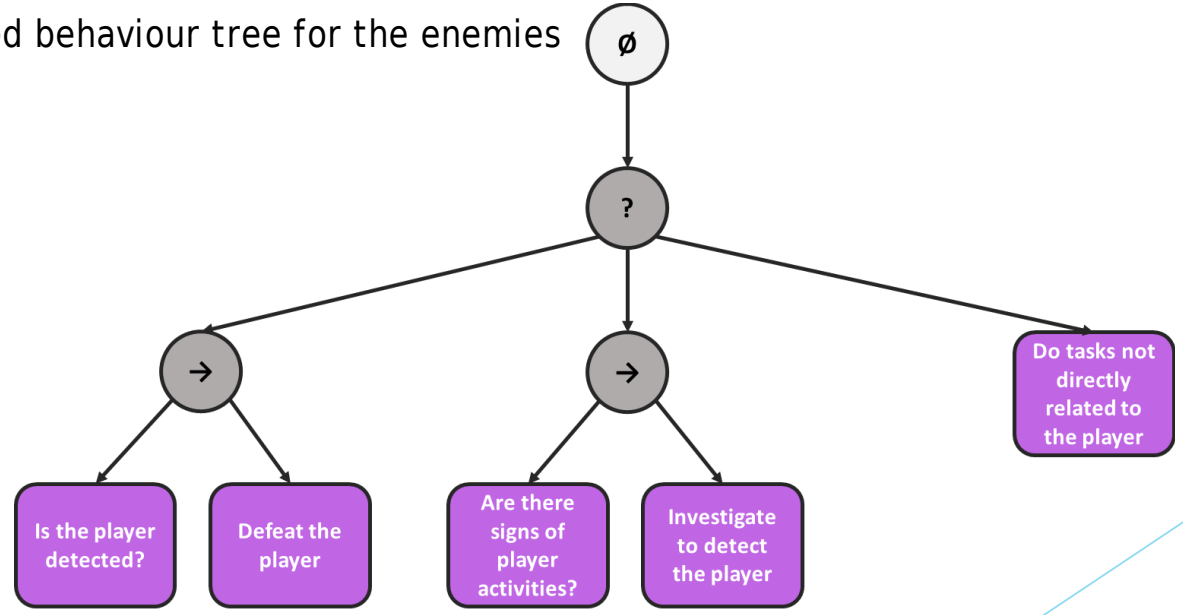




The enemies' individual reasoning and decision-making (I)

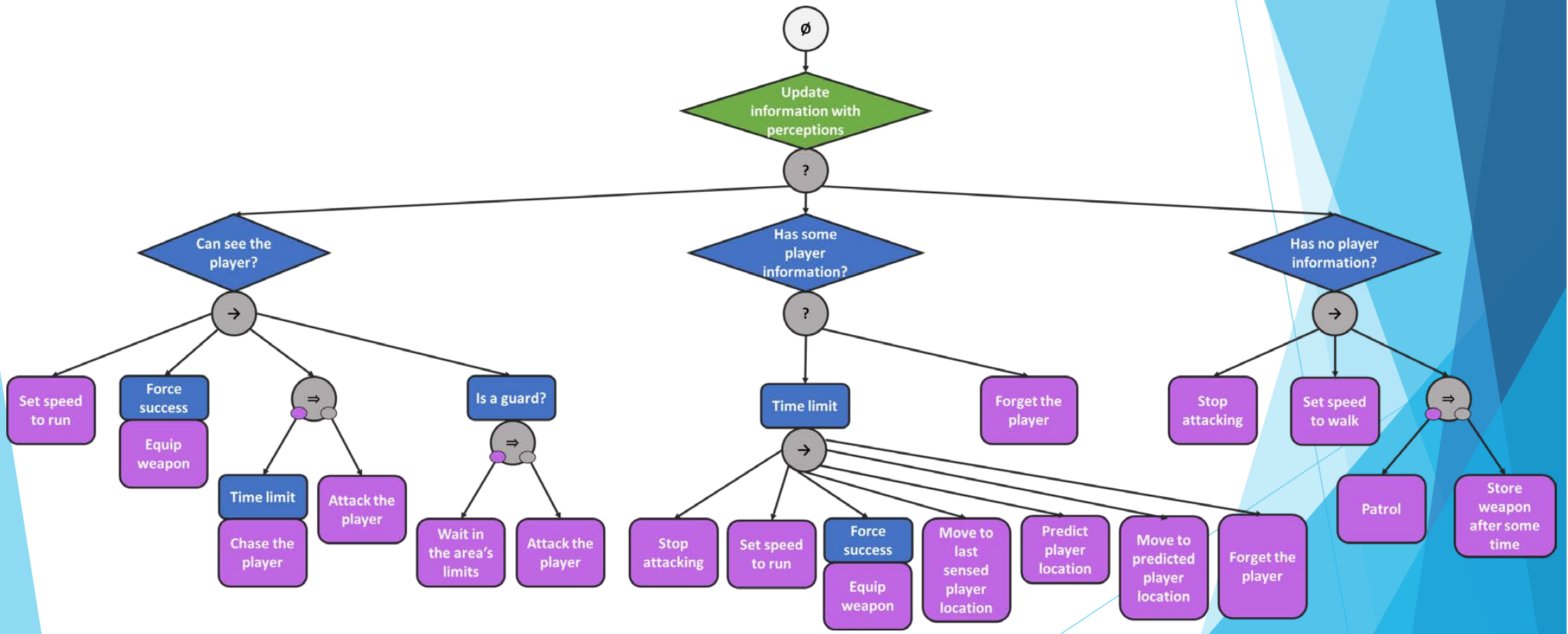
- The individual reasoning and decision-making is modelled with a behaviour tree. It is a system updated in every frame that performs checks and action-based tasks, some of which take multiple frames to complete.
- Conceptually, enemies can be in three different situations or states:
  - Defeat-the-player state, only possible when the player character is detected.
  - Suspicion state, in which the enemy investigates signs in the environment that may be related to the player and lead to detection.
  - Passive state, in which the enemy has no relevant information about the player, but can patrol to try to find the character or some clues.

Simplified behaviour tree for the enemies



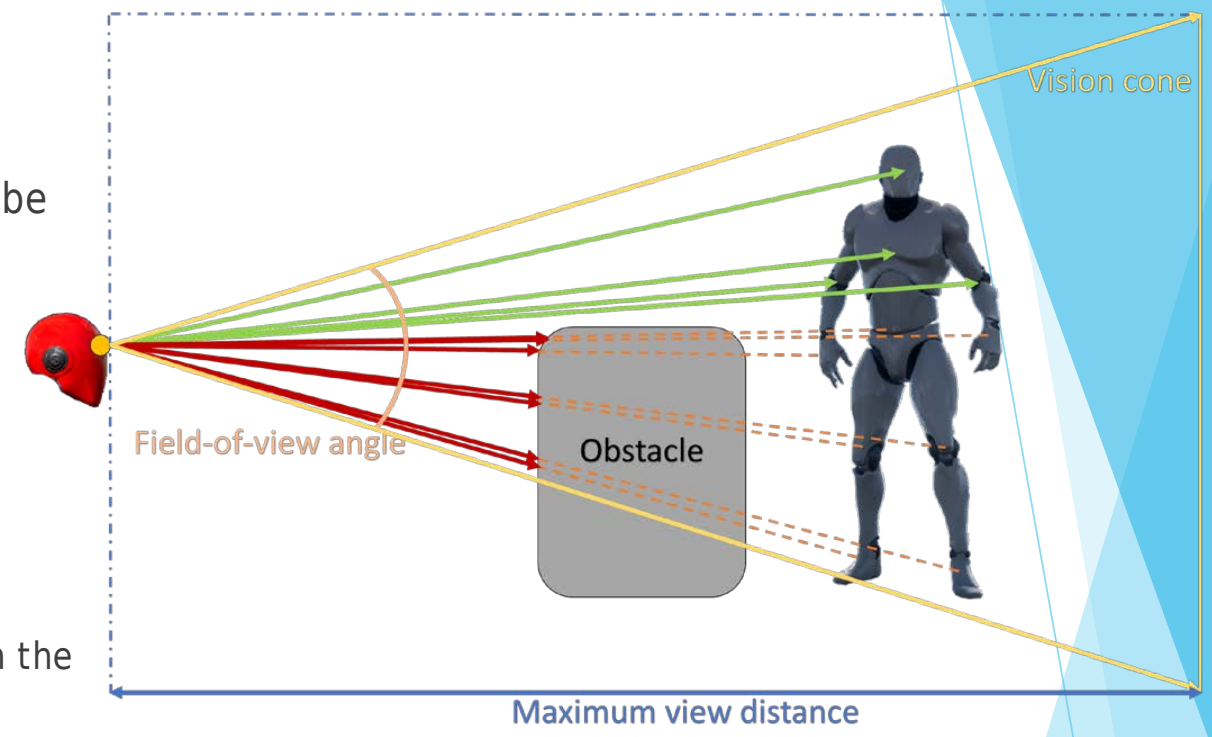
The enemies' individual reasoning and decision-making (II)

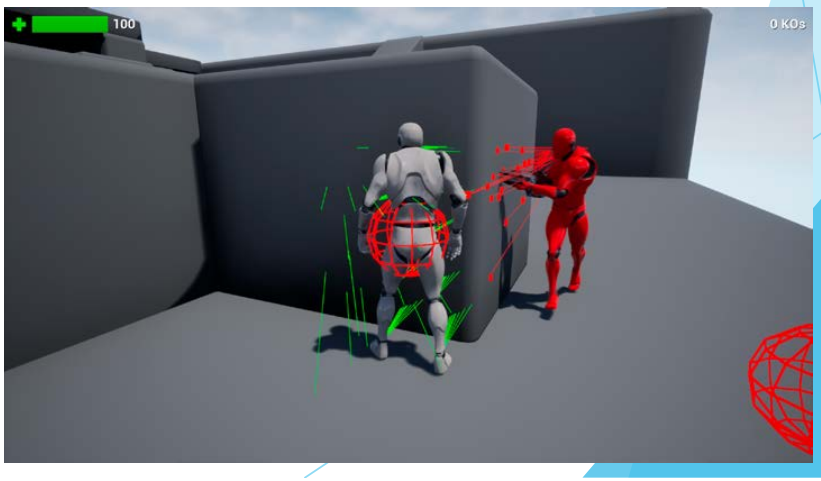
Complete behaviour tree for the enemies



Visual detection (I)

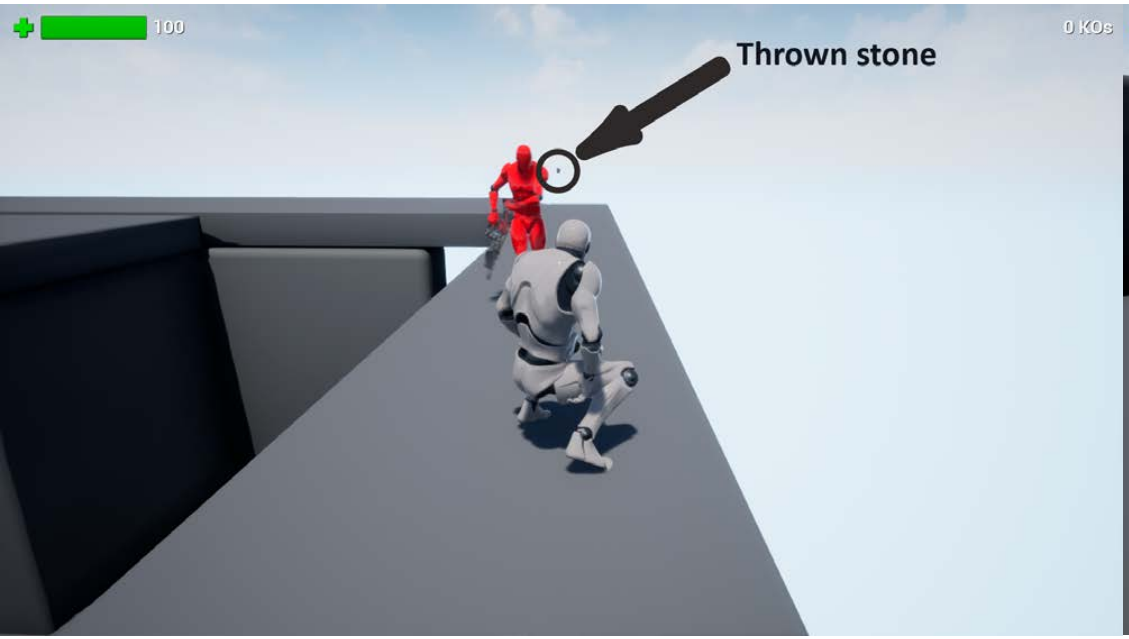
- Three different game elements can be detected with vision:
  - The player character.
  - Thrown objects in movement (e.g. stones).
  - The bodies of unconscious agents.
- Player character's visual detection requirements:
  - The player character must be within the maximum view distance range.
  - The player must be within the cone-shaped region defined by the field-of-view angle.
  - At least one ray trace from the enemy's eyes to a body part of the player character (e.g. head, thorax) must hit the character, not an obstacle.

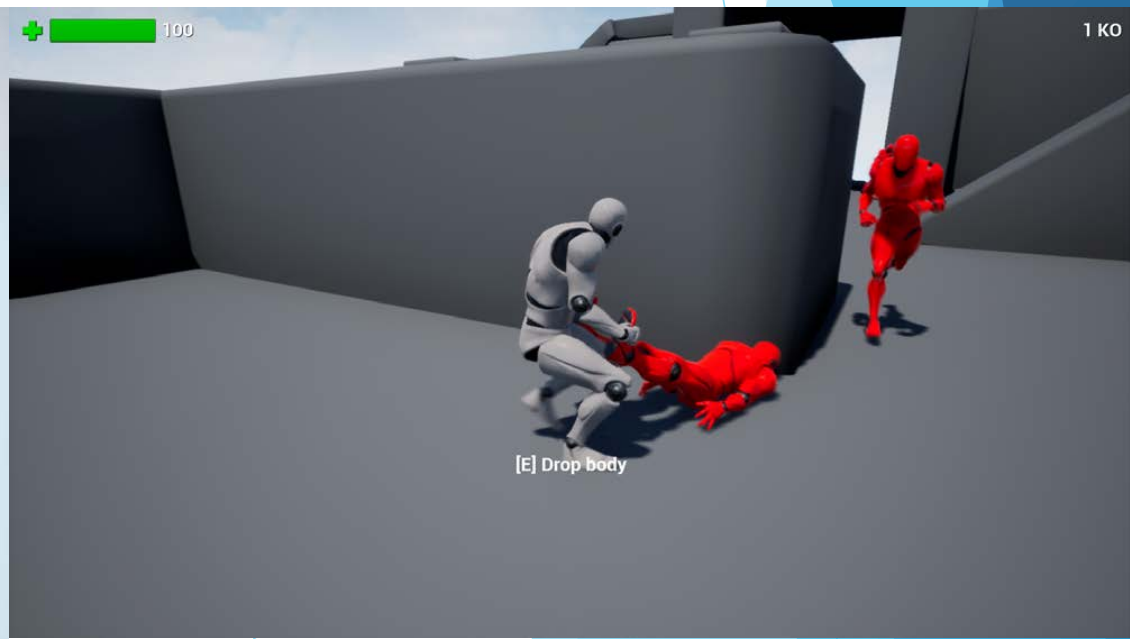




Visual detection (II)

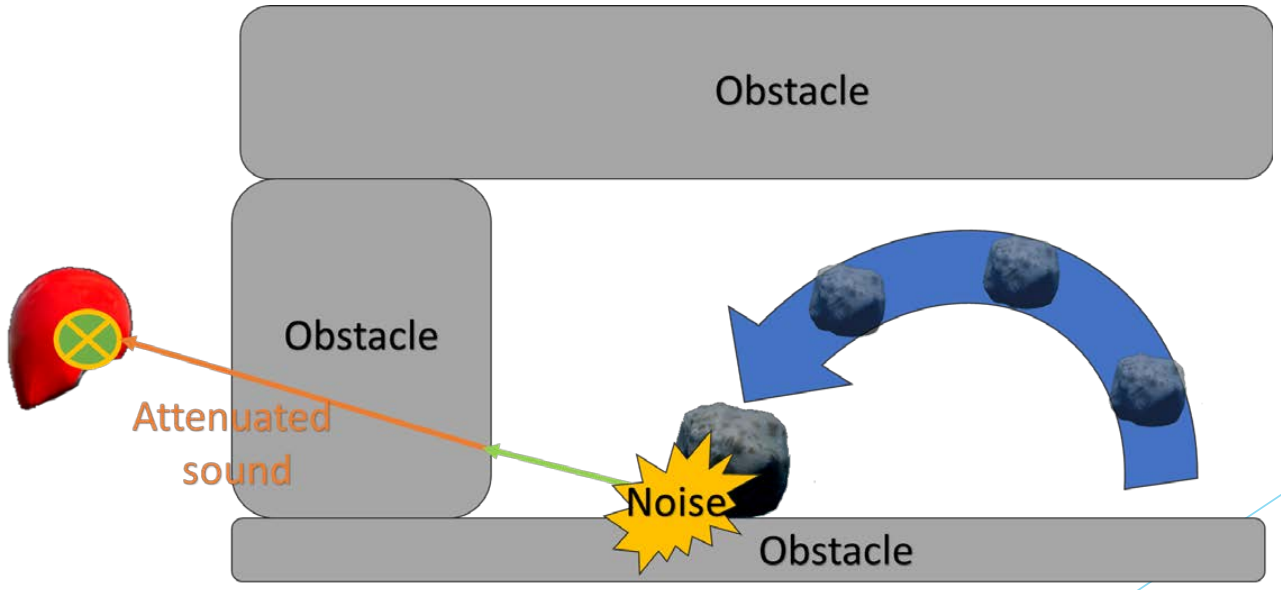
- The visual detection process of objects and enemies' bodies is similar to that of the player character, but with some particularities.
- A throwable object must have a high speed to be interesting for the enemy; stationary objects are irrelevant.
- When enemies see an object in the air, they analyze its inverted forward vector to estimate where has it been thrown from, and investigate that zone.
- A body is only interesting in two different scenarios:
  - It is seen in movement: the player character is assumed to be next to the body.
  - The body's agent had not yet been reported to be unconscious: the player character might be still in the zone.





Auditory detection

- The player character makes noises when running (footsteps) and jumping (landing sound).
- Thrown objects also produce noises when they hit something.
- Each sound has a base loudness, and engine-driven attenuation calculations are used to determine the final volume with which every enemy would hear a sound.
- Enemies receive sound stimuli via an interruption event, only if the destination volume makes the sound audible (i.e. the sound does not come from too far away nor it is highly attenuated by obstacles).
- The stimuli is composed of the type of sound and the estimated source location, which can be imprecise if the sound comes from far away.

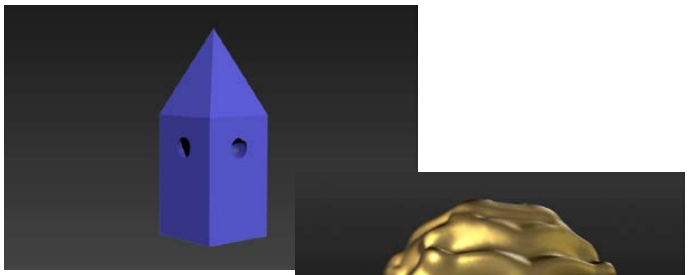


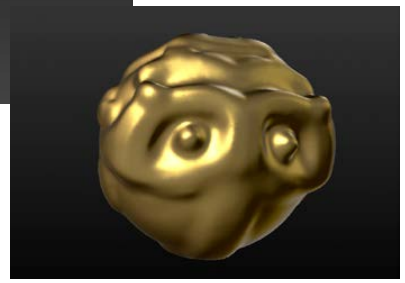
The game's visual appearance and audio

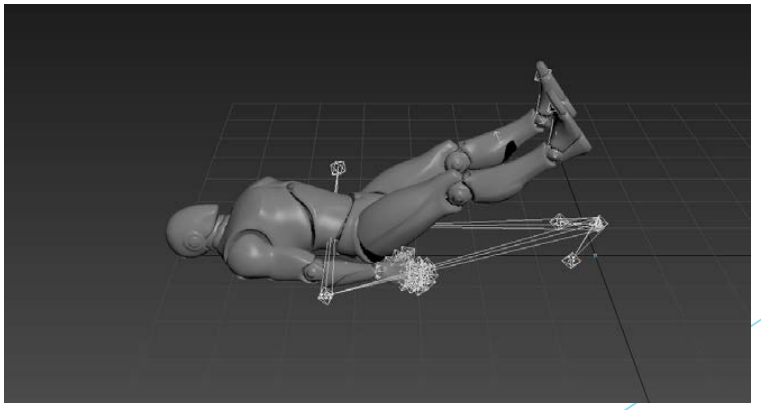
- Most of the used visual and audio assets were obtained from free Unreal Engine samples, but a few of them were manually created.

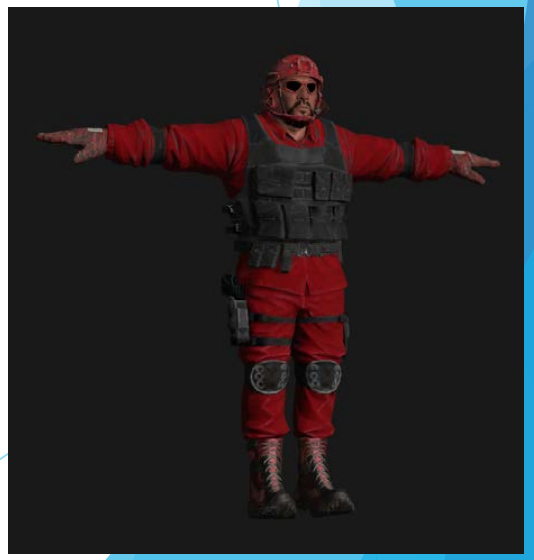
- Content from samples:
  - Temple level (without four-elements-based decoration)
  - Thematic decoration elements (lava, waterfalls, etc.) that were manually added to the temple
  - Main characters' meshes and most animations
  - Most textures, materials and visual effects
  - Sounds and music

- Content designed for the game:
  - Monolith and orb meshes (in 3ds Max and Sculptris)
  - Some animations: object-throwing, body dragging, etc. (in 3ds Max and Unreal Persona editor)
  - Secondary character's mesh (in Fuse)
  - Some textures and materials, e.g. health bar's plus sign (in Photoshop and Unreal Material editor)









Part 3 - Conclusions

Conclusions

- The main objective of the project, which was to create a stealth game with artificial intelligence-powered cooperating enemies, was successfully completed.
- All 7 planned feature modules were completed in time, i.e. all the scheduled tasks were done. 9 videos made during the development process demonstrate this.
- Extra features were added, namely a cinematic introductory video that plays when the game starts, to excite the player and narrate the background and story of the characters.
- Behaviour trees proved to be a flexible system to model non-player character decision-making. It was easy to expand them with new abilities and types of decisions, and they are scalable to different multi-agent system configurations without destabilizing performance.

Possible improvements and expansions

- The gameplay would be enriched with new abilities for the player character, such as being able to temporarily disguise as a knocked-out enemy, to avoid being detected.
- Darkness would be a stealth mechanism if the enemies' visual detection took into account the light intensity in the body parts of the player character. It would be required to do real-time checks of point lights and directional lights to measure their light contribution to the character body parts.
- The game would last longer if more temple zones or even different temples were added.
- The game would look more unique with a greater proportion of custom visual and audio assets.
- The cinematic video would be more immersive if it was produced using specific cinematic tools, such as Unreal Sequencer, to control cameras and animations in a more exhaustive way.

Part 4 - Demonstration

Part 5 – Questions and answers