Maze Chase

Marcos Vázquez Rey

Motores III, ESAT 2017

Table of Contents

1. Game Intro

1.1 Game description

Maze Chase is a game demo that could fit into the *horror game* genre, where the player must struggle to survive a danger that exceeds his capacity to defend, and so he must sneak and avoid the danger, instead of fighting it like it's common in other styles of games.

Our hero spawns inside a labyrinth where (so says the legend) roams a vicious minotaur that feeds upon human flesh. There is an exit somewhere marked by a unique column, and the path is sprinkled with doors and switches that must be activated in order to escape.

The vicious minotaur is capable of both seeing our hero across a certain distance towards his line of sight, and hearing his footsteps and screams. He will haunt the player and chase him for a certain period of time, before going back to patrolling the maze halls.

Our hero has a limited mental health, that will be gradually diminished when running (due to exhaustion), screaming or being screamed at.
The maze will be considerably harder to navigate if he's dizzy, exhausted o panicking, so watch out for your nerves.

Your only goal is to get out of there alive. Now, run.

1.2 Influences

This game takes influence from many other games , particularly some in the *horror* genre. Some of those titles would be *Silent Hill*, *Resident Evil* or *Slender*.

Nevertheless, some of the most characteristic features in the genre have been lowered in intensity, creating something a little bit more action-oriented (such as the field of vision and the player speed of movement).

Here's some of the elements taken from the games previously mentioned:

- **Reduced lightning.** Lightning is very important when creating a dark and claustrophobic atmosphere. The overall lightning has been obscured and the player must rely on his torchlight to clearly see what's around him.

- **Reduced movement.** Forcing the player to move slowly creates a sense of defenselessness.

- **Dizziness.** Playing around with some camera shaking and post-processes helps to transfer the nervousness from the character into the player.

- **Ambient music.** Music is always important in gaming, and this case is no exception. This genre calls for a track that's purely dark ambient with low pitch notes, or plays slow and separated notes, or just plays in a slow tempo. It prevents the player from relaxing fully and creates awareness of danger behind every corner.

- **No scape.** The minotaur moves as fast as the player does, so he must compromise and get dizzy in order to gain a little speed and be able to run away from a certain death.

2. Included features

These are some of the most remarkable features included in this demo:

Menus & user interface
This demo uses a system of *User Widgets* to display information to the player and provide for interaction. This will be detailed further down this document.

Assets
Meshes, textures and animations, as well as other assets obtained via free and unlicensed sources, make every building and character distinct from each other and provide additional visual feedback to the player.

FMOD integration
All sfx and music uses the *FMOD* middleware to control the volume and effects for the sound output.

Sound & SFX
This includes character sounds for roaring, screaming, jumping and others. As well as specific sounds for the doors.

Ambient Music
Scary

Lightning
Both in the form of torches attached to the maze walls, and a torchlight attached to the player's head that can be switched on and off.

AI Minotaur
The baddie of this demo makes use of sensing components and behaviour trees to act on his own.

Procedurally generated maze
The labyrinth generates procedurally and automatically when the level is loaded. Further explanation on this can be found in the *designer's guide* section of this document.

Complete animations
Characters use animations to provide visual feedback for their actions. These animations also trigger events to affect the overall world (e.g. emitting a sound).

Post-processes
These get triggered as the hero's nervousness increases, hampering navigation.

3. User interface & player interaction

3.1 UI Widgets

As previously stated, this demo makes use of a series of interconnected widgets to provide information and interaction to the player.
Widgets contain several buttons that trigger game actions.
There is only one widget at a time on screen, so when one is triggered all the other ones are removed from viewport.
Depending on the game map they can also be of different types:

Main Menu
This widget appears as the *MainMenu* map is loaded. It uses a dynamic material generated from a spritesheet, creating the illusion of a video playing in the background.
Even though the video quality is pretty crappy, it serves as a demo of how such an effect could be achieved.
It also contains a button that leads to the main game, an another one to exit the program.

In-game HUD
This widget appears in the main game. It provides information about the characters:
The *Sanity* bar represents the mental health of our hero.
The images that appear next to it represent the minotaur's level of awareness. They will show in a gray scale if nothing is happening, and colored otherwise.
The minotaur face means that the player is being chased, the eye means that he's within line of sight with the minotaur, and the ear means that the player has just been heard and detected.

Pause Menu
This simply pauses all action in the game and shows a simple widget that says so.

Game Over
Shown when the minotaur catches the player. It displays the survived time and immediately fades to black onto the *Main Menu*.

Game Won
Shown when the player reaches the exit. It displays the escape time and immediately fades to black onto the *Main Menu*.

3.2 Game mechanics

3.2.1 Player controls

The player interacts with the game by controlling our hero and actions can be triggered through the following mechanics:

Look around
The player can move the mouse to look around and make the camera rotate around the main character.

Walk
The main character can be controlled with the *ASDW* keys to make him walk around. The direction of movement will be relative to the camera forward, and not towards where the player is headed.
Walking makes a moderate amount of noise and the minotaur can detect our hero by his footsteps.

Run
The *left shift* key will make the main character sprint and gain a 50% speed bonus while it's active.
Note that while sprinting the character will get tired and this will reduce his *sanity* levels.
Also, running makes a quite a bit of noise and the minotaur will detect our hero if he sprints anywhere near him.

Sneak
The *left ctrl* key will make the main character sneak and advance at 50% velocity from walking.
Sneaking makes a very low level of noise and it's useful for passing near the minotaur without being heard.

Jump
The *spacebar* makes our hero jump and emit a noise. Unlike the footsteps, this sound cannot be heard by the minotaur.

Push Buttons
The player can push buttons that temporarily open a set of doors by facing a *DoorButton* and pressing the *E* key.

Scream
Pressing the *Q* button makes the character scream in fear. This makes a hell of a noise and will be heard by the minotaur pretty much anywhere inside the labyrinth.

The player’s sanity levels will also decrease while he’s screaming.

Flashlight
Pressing the *P* button will switch the player’s flashlight on and off. This can be useful for examining the environment or creating a darker atmosphere.

3.2.2 Other mechanics

Sanity
The hero’s sanity level will decrease if he runs, screams or is screamed at. This will apply a camera shake, a pixelating post process and it will also steer him sideways when moving forward. All of this happens proportionally to the sanity meter.
Sanity will recover over time and these effects will gradually dissapear.

4. TDD

4.1 Target Specs

Following are the specifications the game has been developed and targeted towards. So this demo is guaranteed to work on the following setup:

- PC Windows 10
- Graphics card Nvidia GTX960
- 8GB DDR4
- Intel Core i7-6700HQ 2.6GHz
- Unreal Engine 4.13.2
- DirectX 11 shader model
- FMOD Studio 1.09.01
- Visual Studio 2015 or C++ redistributable packages installed.

4.2 Unreal Engine features

Other than the basic classes and blueprints. Here's some of the additional Unreal Engine features used during the development of this demo:

- Point lights for torches.
- Particle effects of fire.
- Materials on walls, ceiling, ground and other actor elements.
- Camera shake and post processes for sanity effects.
- Level sequence for fading to black at the end of the game.

4.3 FMOD features

This demo makes use of the FMOD audio middleware, using a single master bank for holding all the music & sound effects.
The music is implemented as 2D sound with no distance attenuation, while the character sounds are played at the character positions and fade away with distance.
Some effects, such as the minotaur roar or footsteps make use of scattered sounds for adding variety and playing a random effect from a set of sounds.
The volume for this effects also varies in a degree to add a sense of realism.

4.4 Code classes

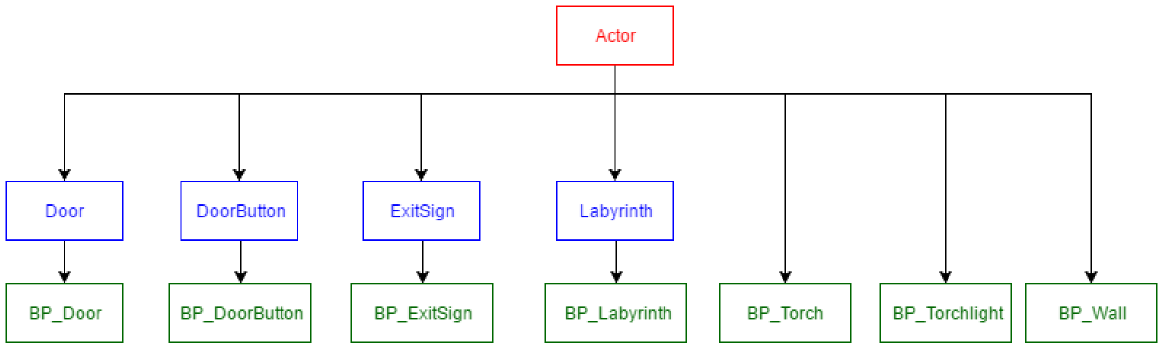
This demo is first and foremost written in C++, although it uses some blueprint implementations in cases where the C++ code would become too cumbersome, visual aid is needed or final inheritances in blueprint are convenient.

The following sections graphically depict different categories of elements implemented and their hierarchy.
Also, clarifications on the inner workings of the classes are appended here.
Nevertheless, the source code should be consulted for more details, since it's heavily commented and will help better understand how these classes work.

For the inheritance pictures, the legend is as follows:

* Red boxes represent an engine base class.
* Blue boxes mean an implemented C++ class
* Green boxes mean a blueprint class that inherits from a C++ one.

4.4.1 Actors



Door
A wall-like actor that interacts with *DoorButton* to create the opening/closing doors mechanic.
It includes an FMOD sound class and an instance to play a grinding effect when moving.
It also includes several *TimerHandles* for managing automatic opening and closing.
Check the designer's guide for more info on how to use these two components together and create cool level design.

DoorButton
This is the other part of the deal**.** Door buttons can be interacted with to open a predefined set of doors that will also close automatically after a specified time.

ExitSign
This actor has a collider attached that serves as condition for player victory. When the player character overlaps this Actor, the game is beat and the Widget_GameEnd is added to viewport.
The game will go back to the main menu shortly after that.

Labyrinth
This actor procedurally generates a whole maze around it when a map is loaded. Several actor references must be added to it to specify the meshes that will be used doors, torches, the minotaur enemy… etc.
Several other parameters can be specified to customize the generated maze, such as size and complexity.
Note that a new maze will be generated every time a map with this actor is loaded, and if there's any other present it won't be deleted.
This means that if a labyrinth is generated, the Labyrinth actor must be deleted from the map not to have 2 superposed labyrinths the next time the map opens.
Alternatively, you can delete the labyrinth components (everything under the "Maze" folder, such as actor walls and torches), and a new one will be generated on map reload.

Normally, the map would be composed of several layers each one having lightning, gameplay, or art elements, but since the level components are all generated on the fly, this makes it poorly practical to distribute them in layers, and it uses a single persistent map file.

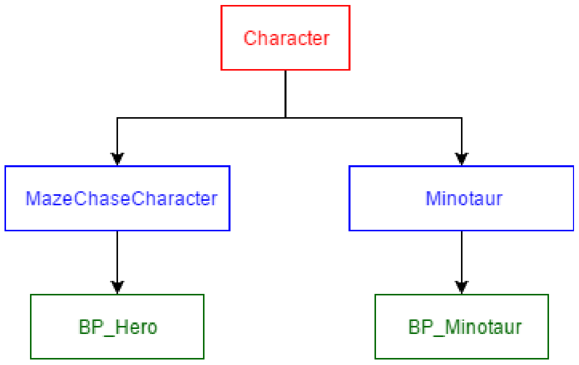
Torch
Torches are just a rectangular mesh with a red point light attached to its tip that help illuminate the maze.

Torchlight
A cube with a spotlight that is attached to the player's forehead and can be switched on and off.

Wall
A static mesh that is used to construct the labyrinth walls.

4.4.2 Characters

Maze Chase is a 1vs1 game that contains only 2 characters. They are both created in C++ and inherited in a blueprint class.

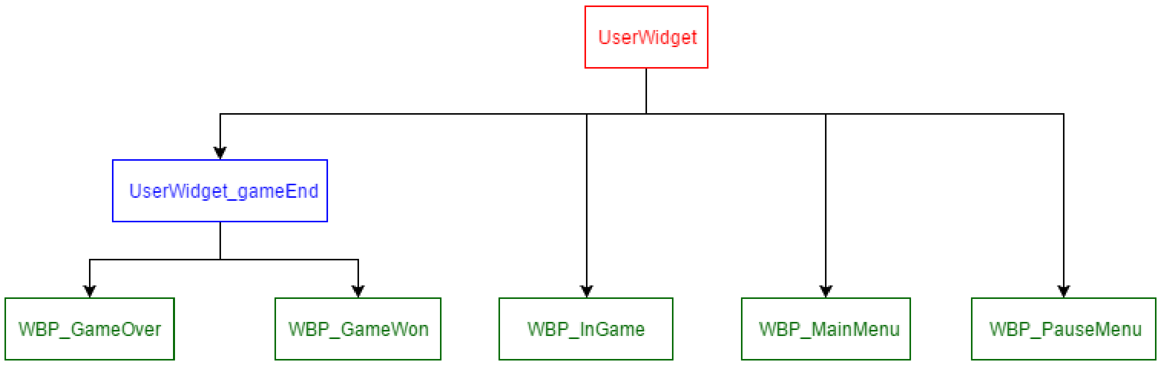


Hero
This is the player-controlled character that must escape the maze in order to win the game.
The hero has attached a noise emitter component that makes sound audible by the minotaur at certain distances and is triggered on animation notifies (it emits sound on certain animation key frames).
The blurring post-process is applied directly to the hero's camera

Minotaur
This is the main and only villain in *Maze Chase*. It will search the player around the maze and chase him to death.
The minotaur has a pawn sensing component that allows him to see the player pawn and hear the sound he emits.
Its behavoir is controlled by a behaviour tree and several BTService. The minotaur platrols between several points placed randomly inside the maze. If he hears or sees the player, it will chase him through the maze for a few seconds, and then go back to his patrolling duties.
If the minotaur touches the player, it's player dead and game over.

4.4.3 User Widgets

This set of widgets display information to the player inside the game.



Widget_GameEnd
A simple widget that displays a message and activates a timer to go back to the main menu.

Widget_InGame
This delivers information inside the main game about the player's exhaustion level and the minotaur detection status. The eye icon means that the minotaur has seen you, whilst the ear icon means that the minotaur has heard a noise you've made.

Widget_MainMenu
This widget showcases a dynamic material background on how a short video could be made to play within a menu. It also serves a nexus before the main game. It allows the game to start and quit.

Widget_PauseMenu
A simple widget that pauses the game action and shows when the game is paused.

4.4.4 Other classes

AnimNotifies
Several Blueprint animation notifies are created to play sounds and alter the game mechanics.

Level Sequence
A level sequence fades the screen to black when is game is over (either won or lost).

BlendSpaces
In addition to animations, 1D blendspaces are created to transition between still and walking/running states in the hero and minotaur.

Animation Blueprints
These are used to trigger animation states in the characters.

5. Roadmap

This section lists some of the features and game mechanics that could be added to the game if enough time were available:

Better lightning.
Ambient could be tweaked with some sort of atmospheric fog .
Torches could use a dynamic lightning to better simulate fire.

Better materials.
The walls need improving to enhance the sense of deepness through light reflection.

Particles.
Fog, fire and other particles would improve the immersion and overall graphical aspect.

Way more animations.
More animations for all the characters.

More minotaurs
The game could be made more phrenetic by adding a custom amount of minotaurs and a medium to slow them temporarily.

Traps
In the same way, adding traps and other game elements would alleviate the monotony of just running around dodging the minotaur.

Decorative elements
Horror vacui.

Designer-customizable maze size.
The maze size is not among the variables that can be adjusted by a game designer. This was removed due to the maze generation algorithm being reliable on static memory that must be initialized at compile time, and thus cannot be reallocated later.

6. GDD

6.1 Game rules

You awake in the maze the minotaur is said to roam around. You must go through the labyrinth halls and find the exit before the minotaur finds you.

The minotaur can see you when he's looking at you. If that happens, he'll lock on you and follow after a few seconds after he's lost track of you.
The minotaur can also hear you scream and hear your footsteps.

You can move in 3 different speeds: crouch, walk and run. The faster you move, the noisier you are.

The game ends when you reach the exit of the minotaur touches you.

6.2 How to play

Keyboard controls:

Mouse	-	Look around
ASDW	-	Move around
Spacebar	-	Jump
Left Shift	-	Run
Left Control	-	Crouch
E	-	Fly / Press button
Q	-	Scream
L	-	Turn flashlight on/off

7. Development log

Here's some insight into some of the most remarkable (or hard) things I've learned by developing this demo:

7.1 Problems and solutions

Some problems found and solutions applied:

Procedurally generated maze

A procedurally generated maze that always has a viable solution.
This is built by constructing an internal model made of cells with 4 walls each (walls can be shared among cells). The algorithm sets a start and an end and carves the walls advancing from cell to cell until a solution is found.
Based on this model, actors are spawned then, and doors, torches and other elements are added by checking against to model for walls.

Video on main menu

A simple and not very satisfactory solution has been to add a dynamic material that iterates through sort of a spritesheet to create the illusion of video. A better aproach should be further investigated.

Problems with character meshes and animation

Applying animations and skeletal meshes from different sources proved to be somewhat of a chore. Some rigging was needed.

7.3 Rejected ideas

Some ideas that were tested and rejected:

Automatic door button placement

This idea was rejected because of the inherent complications of making for all the combinations of doors, buttons and player position that could lead to a dead end.

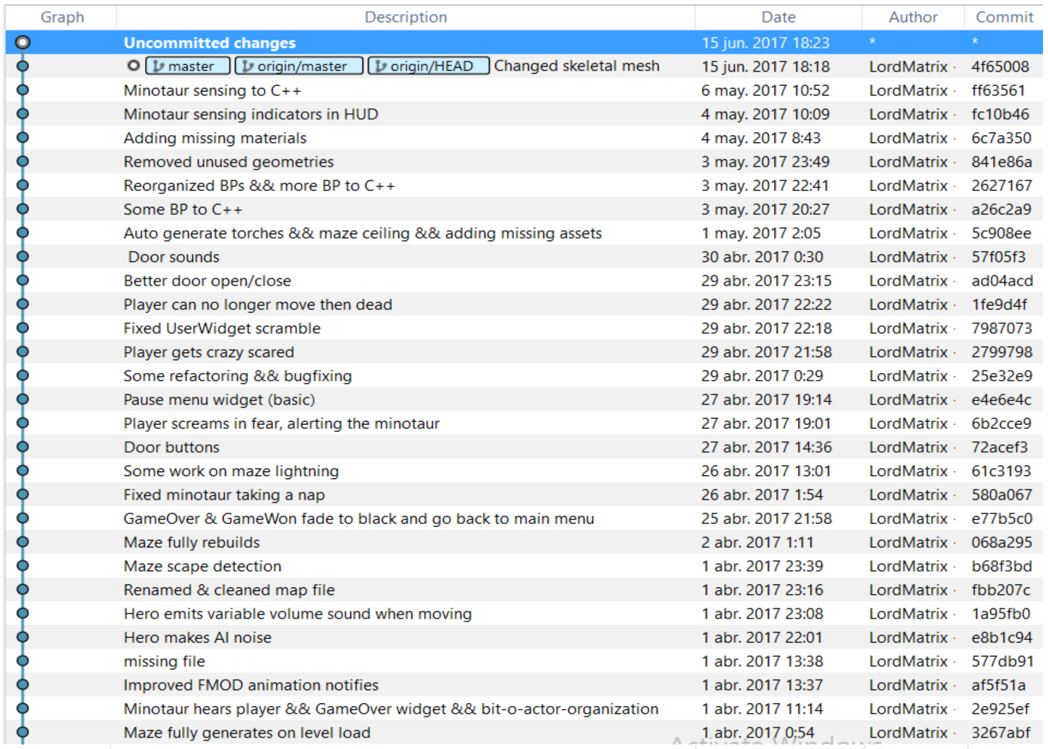
The player could easily get trapped in a hall with a door and no button that opens it nearby. So now the buttons and associated doors are set by hand.

Torches with emissive particles

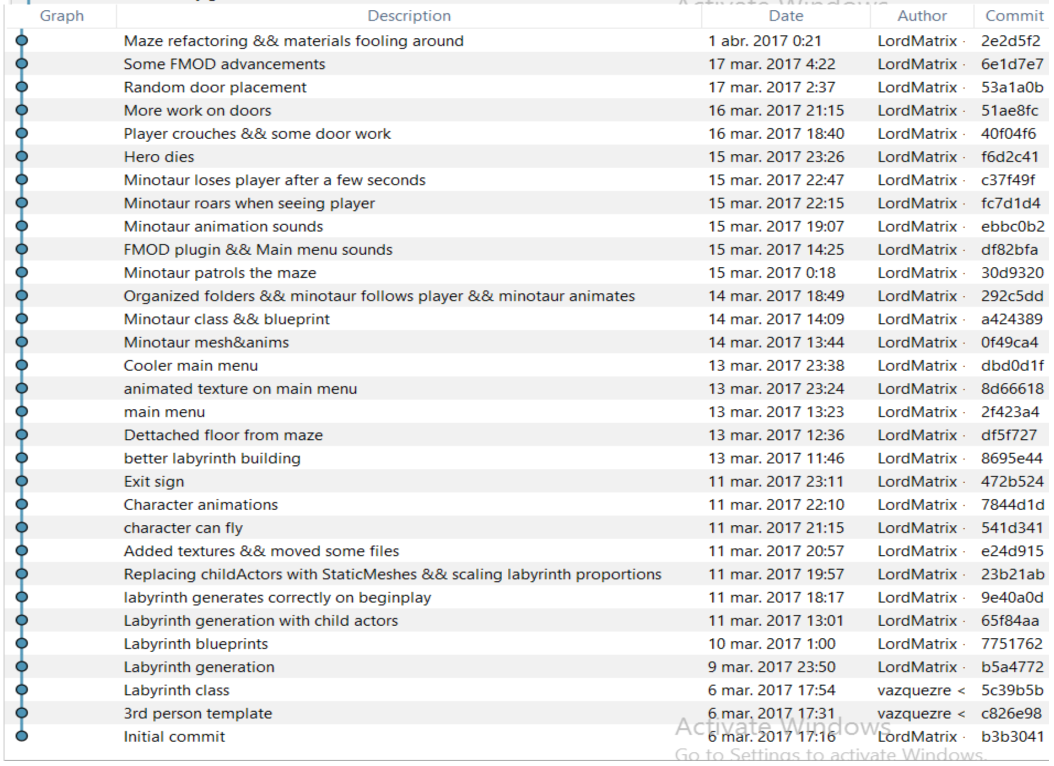
Torched were made with emissive light particles in mind. This was removed after a pesky bug caused light leaking though the walls and floor, totally breaking immersion.

7.2 Log

This page shows screenshots of the commit history for the repository that contains the project, serving as development log of achievements over time.



Graph	Description	Date	Author	Commit
	Uncommitted changes	15 jun. 2017 18:23	*	*
	master origin/master origin/HEAD Changed skeletal mesh	15 jun. 2017 18:18	LordMatrix ·	4f65008
	Minotaur sensing to C++	6 may. 2017 10:52	LordMatrix ·	ff63561
	Minotaur sensing indicators in HUD	4 may. 2017 10:09	LordMatrix ·	fc10b46
	Adding missing materials	4 may. 2017 8:43	LordMatrix ·	6c7a350
	Removed unused geometries	3 may. 2017 23:49	LordMatrix ·	841e86a
	Reorganized BPs && more BP to C++	3 may. 2017 22:41	LordMatrix ·	2627167
	Some BP to C++	3 may. 2017 20:27	LordMatrix ·	a26c2a9
	Auto generate torches && maze ceiling && adding missing assets	1 may. 2017 2:05	LordMatrix ·	5c908ee
	Door sounds	30 abr. 2017 0:30	LordMatrix ·	57f05f3
	Better door open/close	29 abr. 2017 23:15	LordMatrix ·	ad04acd
	Player can no longer move then dead	29 abr. 2017 22:22	LordMatrix ·	1fe9d4f
	Fixed UserWidget scramble	29 abr. 2017 22:18	LordMatrix ·	7987073
	Player gets crazy scared	29 abr. 2017 21:58	LordMatrix ·	2799798
	Some refactoring && bugfixing	29 abr. 2017 0:29	LordMatrix ·	25e32e9
	Pause menu widget (basic)	27 abr. 2017 19:14	LordMatrix ·	e4e6e4c
	Player screams in fear, alerting the minotaur	27 abr. 2017 19:01	LordMatrix ·	6b2cce9
	Door buttons	27 abr. 2017 14:36	LordMatrix ·	72acef3
	Some work on maze lightning	26 abr. 2017 13:01	LordMatrix ·	61c3193
	Fixed minotaur taking a nap	26 abr. 2017 1:54	LordMatrix ·	580a067
	GameOver & GameWon fade to black and go back to main menu	25 abr. 2017 21:58	LordMatrix ·	e77b5c0
	Maze fully rebuilds	2 abr. 2017 1:11	LordMatrix ·	068a295
	Maze scape detection	1 abr. 2017 23:39	LordMatrix ·	b68f3bd
	Renamed & cleaned map file	1 abr. 2017 23:16	LordMatrix ·	fbb207c
	Hero emits variable volume sound when moving	1 abr. 2017 23:08	LordMatrix ·	1a95fb0
	Hero makes AI noise	1 abr. 2017 22:01	LordMatrix ·	e8b1c94
	missing file	1 abr. 2017 13:38	LordMatrix ·	577db91
	Improved FMOD animation notifies	1 abr. 2017 13:37	LordMatrix ·	af5f51a
	Minotaur hears player && GameOver widget && bit-o-actor-organization	1 abr. 2017 11:14	LordMatrix ·	2e925ef
	Maze fully generates on level load	1 abr. 2017 0:54	LordMatrix ·	3267abf



Graph	Description	Date	Author	Commit
	Maze refactoring && materials fooling around	1 abr. 2017 0:21	LordMatrix ·	2e2d5f2
	Some FMOD advancements	17 mar. 2017 4:22	LordMatrix ·	6e1d7e7
	Random door placement	17 mar. 2017 2:37	LordMatrix ·	53a1a0b
	More work on doors	16 mar. 2017 21:15	LordMatrix ·	51ae8fc
	Player crouches && some door work	16 mar. 2017 18:40	LordMatrix ·	40f04f6
	Hero dies	15 mar. 2017 23:26	LordMatrix ·	f6d2c41
	Minotaur loses player after a few seconds	15 mar. 2017 22:47	LordMatrix ·	c37f49f
	Minotaur roars when seeing player	15 mar. 2017 22:15	LordMatrix ·	fc7d1d4
	Minotaur animation sounds	15 mar. 2017 19:07	LordMatrix ·	ebbc0b2
	FMOD plugin && Main menu sounds	15 mar. 2017 14:25	LordMatrix ·	df82bfa
	Minotaur patrols the maze	15 mar. 2017 0:18	LordMatrix ·	30d9320
	Organized folders && minotaur follows player && minotaur animates	14 mar. 2017 18:49	LordMatrix ·	292c5dd
	Minotaur class && blueprint	14 mar. 2017 14:09	LordMatrix ·	a424389
	Minotaur mesh&anims	14 mar. 2017 13:44	LordMatrix ·	0f49ca4
	Cooler main menu	13 mar. 2017 23:38	LordMatrix ·	dbd0d1f
	animated texture on main menu	13 mar. 2017 23:24	LordMatrix ·	8d66618
	main menu	13 mar. 2017 13:23	LordMatrix ·	2f423a4
	Dettached floor from maze	13 mar. 2017 12:36	LordMatrix ·	df5f727
	better labyrinth building	13 mar. 2017 11:46	LordMatrix ·	8695e44
	Exit sign	11 mar. 2017 23:11	LordMatrix ·	472b524
	Character animations	11 mar. 2017 22:10	LordMatrix ·	7844d1d
	character can fly	11 mar. 2017 21:15	LordMatrix ·	541d341
	Added textures && moved some files	11 mar. 2017 20:57	LordMatrix ·	e24d915
	Replacing childActors with StaticMeshes && scaling labyrinth proportions	11 mar. 2017 19:57	LordMatrix ·	23b21ab
	labyrinth generates correctly on beginplay	11 mar. 2017 18:17	LordMatrix ·	9e40a0d
	Labyrinth generation with child actors	11 mar. 2017 13:01	LordMatrix ·	65f84aa
	Labyrinth blueprints	10 mar. 2017 1:00	LordMatrix ·	7751762
	Labyrinth generation	9 mar. 2017 23:50	LordMatrix ·	b5a4772
	Labyrinth class	6 mar. 2017 17:54	vazquezre <	5c39b5b
	3rd person template	6 mar. 2017 17:31	vazquezre <	c826e98
	Initial commit	6 mar. 2017 17:16	LordMatrix ·	b3b3041

8. Designer guide

These section includes elements modifiable in-game, for game designers to tweak and create levels.

8.1 Create a level

In order to create a level, create a map and then put a BP_Labyrinth in the scene. You can tweak this actor's variables to spawn the maze at your will.

Take into account that a whole labyrinth will spawn for every BP_Labyrinth on the map when it loads. This means that either you'll want a BP_Labyrinth actor in the scene to spawn a randomly generated maze, or a pre-generated maze. If the latter is true, remember to delete the BP_Labyrinth actor after the maze has been generated.

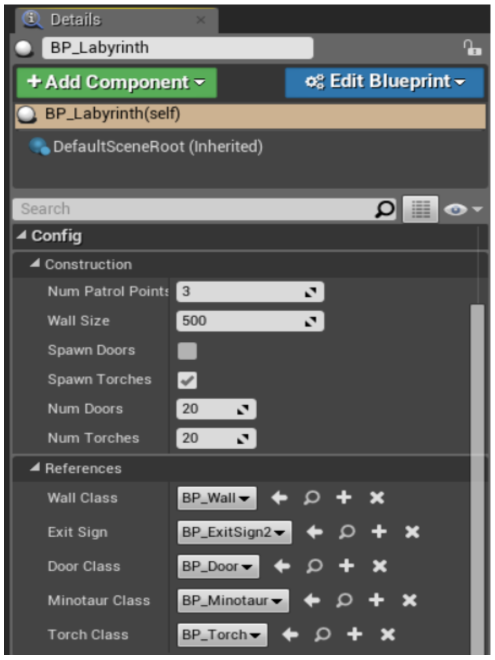
If you want to delete the maze, navigate to the *Maze* folder inside the actor list in the editor, right click → select all, right click → delete. Remember to also delete any player spawn that remains.

You must add a floor to your scene for the maze to be walkable and the game playable. You might also want to add other elements such a ceiling or decorations for pre-generated mazes.

The minotaur and the player will spawn in random places around the maze at floor level.

8.2 Labyrinth

The labyrinth can be customized through the following parameters



Config

Construction

Num Patrol Points	3
Wall Size	500
Spawn Doors	☐
Spawn Torches	✓
Num Doors	20
Num Torches	20

References

Wall Class	BP_Wall
Exit Sign	BP_ExitSign2
Door Class	BP_Door
Minotaur Class	BP_Minotaur
Torch Class	BP_Torch

Num patrol points
The number of minotaur patrol points to be generated.

Wall size
Separation between maze cells and wall actors.

Spawn doors & num doors
Whether or not to spawn doors and how many.

Spawn torches & num torches
Whether or not to spawn torches and how many.

References
Actors that will serve as templates for walls, exit sign, doors, the minotaur and torches.

8.3 Minotaur

The minotaur's main behaviour can be controlled by setting the following group of parameters in their respective components.
Many other variables can be tweaked, but these contain the ones that are intended to be adjusted without breaking gameplay.

Chase Time (root)
For how long the minotaur will chase the player after losing track of his position (the last time he saw/heard him), before going back to patrolling.

Speed&Acceleration (CharacterMovement)
How fast the minotaur moves.

Sensing (PawnSensing)
Range of detection. Sight can be adjusted in length and amplitude of the vision cone.
Hearing can have its threshold altered for sharper detection.