Get Me Out,
A COOP Platform Video Game For The 2020 Ubisoft Game Lab

Report written by Robin Leman, McGill ID 260888756

Game made by

Filipe Rodrigues
Rikke Aas
Chicheng Zheng
Robin Leman
as Programmers

Andrea Bleau-Landry
Stéphanie Lafleur
Jennifer Powroznyk
as Artists

and Thomas Buffard
as Game Designer and Team Lead

ABSTRACT

Get Me Out is a stealth puzzle and platform video game developed in 10 weeks on Unity for the 2020 Ubisoft Montreal Game Lab competition. The game tells the story of two bok-choys, generated and made alive by a mad scientist. Their goal is to escape a laboratory while hiding from the scientist. The competition had multiple constraints: the theme "Generation", a cooperative multiplayer, a progression system and the implementation of an artificial intelligence. Starting from scratch, we had to build an original and innovative game in a team of 8 students, composed of designers, programmers and artists. This paper will discuss the creation of *Get Me Out*, its technical aspects, its innovations, designs and the issues we faced. It will focus on the programming side of the project, being myself a programmer in the team.

I. INTRODUCTION

Get Me Out is a 2.5D Platform and Puzzle video game developed on Unity for the 2020 Ubisoft Game Lab. The game has been developed cooperatively by a team of 4 Programmers, 3 Artists and one Game Designer.

The Ubisoft Game Lab is a competition in the format of a "hackathon" or "game jam" where each team have to create a playable prototype of a video game in 10 weeks. The teams are composed of 8 students representing their University. The video game prototype has to follow constraints imposed by Ubisoft. The constraints of the 2020 edition were the theme "Generation", a cooperative (COOP) and progression system, and the establishment of an Artificial Intelligence (AI) in the game.

Throughout the competition we were helped and mentored by two Ubisoft employees. They helped us on the design and technical side of our game, allocating 1h per weeks for us.

Get Me Out is a 2-player COOP game where each player controls a bok-choy, generated and made alive by a mad scientist. The goal of the players is to solve puzzles while hiding from the scientist to escape the lab. Throughout the levels, the bok-choys gain abilities helping them progress and finish the different levels. The first ability is to grab objects or players, the second one is to throw grabbed objects or players.

We worked on the game for 10 weeks, starting January 17th. We had to work from scratch, find first the design of the game and then develop it on the engine of our choice. The game loop of the game is as follows: the 2 players start a level, gain an ability like grabbing or throwing objects and use these abilities to solve puzzles and finish the level. A loosing state is reached when both players die. At this state, the players respawn to the last checkpoint or the beginning of the level, each game element being reinitialized. A dying state is reached when one player falls in a gap or is caught by the scientist. In a dying state the player is prisoned in a bubble and the second player has to revive him by popping the bubble.

To develop the game, we used the Unity Engine along with other collaborative tools like GitHub or Google Drive.

We were particularly careful of the "three Cs", respectively the Camera, the Controls and the Characters. We established our main mechanics using physics-based technics. For example, we implemented a mechanic to grab objects using Hinge Joints. We implemented both online and local multiplayer. The networking was supported by the Unity Extension "Photon". An Artificial Intelligence was developed in our game with a State Machine, playing the role of the scientist searching for the players.

Finally, it is necessary to remind the context in which this game was made. The COVID-19 pandemic touched us two weeks before the 10 weeks deadline of the competition. We didn't have access to our equipment anymore following the closure of the Universities and we weren't able to meet anymore. The competition has been cancelled and with all of us facing the different issues the virus caused it left the game in a nearly finished state. We all hope to be able to finish the game once the situation improves.

II. MATERIALS AND METHODS

A. UNITY

The first question, before even knowing the theme and constraints of the competition, was to know which engine was going to be used to develop the video game. Ubisoft narrowed down the decision for us to two game engines: Unreal, in C++, and Unity, in C#.

The engine has been one of the main debates of the competition. After meeting the team, it was clear that we all had a lot more experience in Unity, with a lot of us, myself included, never having touched Unreal or coded in C++. However, it was also the main argument for choosing Unreal. The competition was for us the occasion to learn a new engine and a new programming language: C++, a must for the industry.

After a long discussion, we agreed that learning a new engine and a new language while completing the game was too much for us: we already knew Unity and felt much more comfortable with it. Moreover, C# being very close to Java it was for us easier to use it. Finally, Unity possesses a larger library and documentation than Unreal, with a bigger community to help us solving problems.

We started prototyping the game with Unity, a decision that has been valued a few weeks later when Unity offered Unity Premium to every student. It allowed us to use the engine at its full potential.

B. GITHUB, GOOGLE DRIVE & DISCORD

Another issue was how to collaborate on the game. The team was spread out around Montreal, all of us working on different devices. We had to find a way to collaborate and work at the same time on the Unity project without conflicting with each other, and without only being able to work on the project one at a time. GitHub was the obvious decision, as it allowed us to work together on the project while minimizing merging conflicts.

Our approach on GitHub was to create one branch per person, keeping the master branch for a clean version of the Unity project. Then, for each task and issue, we assigned one or two persons to work on it. Once the task was implemented or the issue solved the assignee could create a pull request. After the pull request had been reviewed by another member, to be sure their version didn't contain any bug and wasn't conflicting with the master branch, it was pulled into master.

Even with this approach we had a lot of issues working together on the project without the versions conflicting with one another. Indeed, if two members were working on the Unity scene at the same time the ".unity" file of the scene was changed for both members: thus the two versions conflicted and couldn't be merged.

We tried multiple times to find an easier way to merge scenes together, without any success. Our solution was to manually resolve every conflict caused in the scene.

To keep track of our organization, planning and every other resources and information, we used Google Drive. It allowed us to share screenshots, demonstration video clips, a schedule and every design documents and design information. It was also the tool the artists used to share prefabs and assets together.

Finally, all the communication was made on Discord. We could organize virtual meetings and answer quick questions.

C. ORGANIZATION

To be the most efficient, we agreed to meet the entire team weekly with our mentors from Ubisoft. This weekly meeting was the occasion to discuss the advancement of the game and plan on what was left to be done. It was also a way to re-discuss the design of the game and make some improvements. The meetings were essential for communication, especially between designers, programmers and artists.

As programmers, we met more frequently: almost on a daily basis. We tried to work together in order to help and learn from each other.

The presence of the mentors was crucial to help us answer our questions and help us make decisions both on design and on programming.

III. DESIGN AND GAMEPLAY

Get Me Out is a 2.5D Platform and Puzzle video game where two bok-choys generated by a mad scientist must escape his lab without getting caught.

We spent the 2 first weeks out of the 10 to brainstorm and find the narrative of our new game. One of the biggest issues was to find an idea that was fun, that fitted to the theme "Generation" while matching the constraints of the competition. We also had to keep in mind the 10 weeks scope of the game. First, we all separately brainstormed main ideas to obtain a collection of 10 game designs; that was the base we worked on. We narrowed down the list progressively, regrouping ideas, throwing some away and improving others [1]. We were finally able to have two main ideas and we decided to vote for the best one. This is how Get Me Out was born.

Get Me Out tells the story of two bok-choys, generated by a failed experience of a mad scientist. Indeed, instead of generated an army of mean bok-choys by making them alive, he generated two cute and very nice bok-choys. The game is thus a 2-player coop game, each player controlling one of the bok-choy. Their goal is to solve puzzles together to escape the lab while hiding from the scientist.

This game idea was selected as it answered perfectly to the two biggest constrains that we had: the COOP and the theme. Indeed, the mechanic of 2 players playing each one a bok-choy implemented the Cooperative constrain: we wanted to build puzzles where COOP was not an asset but a necessity to achieve the end of the level. The theme "Generation" was the narrative of our game: the mad scientist generating two bok-choys to make them alive. The progression system was decided to be abilities that the players would gain through the game. The first ability is to grab objects.

[1] Other envisaged game ideas are listed in the appendix.

The second ability, in Level 2, is to be able to throw the grabbed objects. These abilities were design to help the players cooperate and solve puzzles. The more abilities they have, the more puzzles they can solve. Finally, the AI would be the mad scientist searching for the bok-choys.

The game was chosen to be 2.5D: meaning that the movement of the players will be in 2D along the x and y-axis, but they will be evolving in a 3D environment. This format was chosen because the artists felt more confident with 3D assets but also to make it more aesthetic. It was then needed for the implementation of the AI. Finally, a player movement on a 2D axis was chosen as a design aspect to make the puzzles simpler.

IV. THE THREE Cs: CAMERA, CONTROLS, CHARACTERS

The three Cs, namely the Camera, the Controls and the Characters have been our main focus throughout the competition. Indeed, for the scope of the game, the 10 weeks deadline, and considering our past experience, the 3Cs were the main component of our game; it had to be polished for our game to feel good.

A. CAMERA

The camera is one of the main components of the game. Invisible, it is the element that will make the user experience good. Our main issue was to adapt the camera to the multiplayer.

We chose to keep one single camera for both players in a way that the players will always be in the frame of the camera. Not only the players would always be in the frame of the camera, but they would also be framed by the camera: preventing them from distancing themselves too much. This was a way of keeping the players close to each other and make sure they work together in the puzzles. It emphasized the COOP aspect of the game.

We used the implemented Unity extension "Cinemachine". It allowed us to have a well-functioning camera without any scripting. We used a Target Group Camera. It is a camera holding a list of targets, here our two players, to center the frame on the middle point of the two players. The camera will then zoom and de-zoom on the targets to keep them in the frame according to their distance. We added a proper maximum and minimum zoom for aesthetics and a collider at the maximum de-zoom to always make sure the characters don't go out of the camera frame.

Globally the camera using Cinemachine has been a great success. As the Unity engine was itself handling the scripting it prevented us from bugs and saved us a lot of time.

B. CONTROLS

As our game supported a local multiplayer, we needed two players to be able to play on one computer. Sharing a keyboard has been used for testing but we immediately saw that the most intuitive way to play our game was with a controller. A controller provided the best user experience. We thus adapted the controls of our game on this support.

We chose to keep traditional inputs for our game accordingly to what other famous games did. The goal was to keep the controls intuitive for players to be able to approach our game fluidly and learn quickly how to play it.

Not a lot of inputs were actually needed. The players could only use the joystick to move and one single key to jump.

However, we had to think about implementing the more original input we had; grabbing. The first implementation

was just pushing a key on the controller. When play testing we saw that players were intuitively holding the key to grab objects. As it was their natural input, we changed the grabbing control to holding a trigger. Releasing the trigger also releases the object.

Even if we considered our controls to be the most intuitive, we still needed a tutorial to teach basic controls to none-gamers and grabbing. Because of the time constrain we decided to use User Interface (UI) to communicate to the players how to play. We used the environment of the game, a laboratory, to display the UI, controls and tutorials on virtual computer screens.

C. CHARACTERS

The physics of the player movement was really straightforward as we were dealing with a 2D movement. We only needed two types of movement.

The first one was the horizontal movement, along the x-axis of the game. We used an incremental transform position to move the players in a constant speed magnitude in the direction of the player joystick input.

The second type of movement was the vertical one. In our physics-based game this was represented by a jump. The jump was adding a force of type impulse on the player following the correct input. However, to make the jump feel better we decided to apply more gravity at the end of the jump, to make the players fall faster at the end of the jump [2]. When the player starts to jump, he is pushed upwards by a force. When reaching the maximum height of the jump, gravity is increased to accelerate the fall of the player on the ground. This type of jump has been revealed to be very good: it felt more natural for the players and improved their experience.

An issue that brought the 2.5D format and multiplayer was the collision between players. Indeed, as the movement of the players was constant on the z-axis, we had to find a way to make the players pass in front of each other without colliding. This issue is already present in *Little Big Planet*, Sony, 2008. We thus used the same collision mechanism. It consists in anticipating the collision when the two players are close to each other. If the players are about to collide, we move the faster player on another z-axis. The players can then pass in front of each other without colliding.

Another question brought by the multiplayer was the death mechanic. In a single player game, when the player dies it respawn to the last checkpoint. However here if one player dies, he cannot respawn as it would move the player out of the camera frame. Moreover, instant respawning meant that the two players will probably never die at the same time so loosing would be impossible and the game would become too simple. To solve this issue, we were inspired by the Ubisoft game *Rayman Origins*, 2011. This COOP game implements the same camera as our game, *Get Me Out*. When one player died, he became a bubble. He was not able to do anything except slightly controlling the direction of the floating bubble. The second player then had to pop the bubble to free the player. If the second player died when the other one was in a bubble, the two players would die, and they would both respawn to the last checkpoint. This death mechanic was revealed to be the most entertaining for the players. It both kept the punishment of dying, while keeping hope for the player to be revived by the second one.

[2] “Better Jumping in 4 lines of code”, Board to Bits Game, YouTube, 2017

V. MAIN MECHANICS

A. GRABBING OBJECTS

The main feature of the game is that players can grab objects. The implementation of grabbing was highly inspired by *Little Big Planet*. It is the first ability that the player gains in the game. Grabbing is meant to be the main tool to solve puzzles. For example, to access an upper part of the level they can grab a bottle and move it to the wall to help them jump on it to then jump on the upper part. They can also grab objects to drop them on a button and activate a mechanism. Finally, they can grab players to help them navigate in the level or just annoy them and have fun.

It was essential for us to have one single mechanic to solve the puzzles. First, in the context of the game being only a prototype we only had 10 minutes of gameplay; we could not have a lot of puzzles as the players would not have time to do them all. Moreover, it is easier to teach the player one mechanic and then build on it rather than having plenty mechanics and only using them once. The Grabbing mechanic was the Portal Gun [3] of our game.

The design of the grabbing mechanic was that when the player holds a specific input near an object that can be grabbed (called a grabbable for simplicity) the object would be placed above the player's head and follow his movements. The idea is that the bok-choy grabs the object with his leaves, a place it on the top of his head.

The first implementation of the grabbing mechanic was a script attached to each grabbable that handled the mechanic. The grabbable can be grabbed if the player hits the grabbing input and the distance between the player and the object is in a specific radius. Once these conditions are met, the transform position of the grabbable is moved to the position of an empty Game object, above the head of the player. It then made the object a child of the player game object, allowing it to follow its movements.

This first implementation caused a lot of issues. The first one being that the script was attached to each grabbable: it would have been more efficient to have one single script on the player. This caused the bugs of players being able to grab an object that is already grabbed by the other player. Changing the transform position caused the issue of not properly rotating the object on the correct position. The grabbable would then collide with the players. Finally, the ungrabbing process caused bugs. First it consisted on applying an upward vector force on the object and unchilding it from the player. As the game is 2.5D, the z-axis of the players was fixed, but the objects could with the vector force be thrown in a different z-axis than the players, which would cause the loss of the object.

Because of these issues we had to totally rethink the grabbing mechanism. We most of all wanted the grabbing to be more physics based; less robotic and mechanic. After searching the Unity Library for solutions and with the advices of our Ubisoft mentors we decided to reconstruct the grabbing mechanism using Unity's Hinge Joints. Hinge Joints simulate a joint linking two rigidbodies: here respectively the player and the object. We changed the scripting to have only one script attached to each player. This script was supported by a trigger collider around the player, representing the radius in which the player could interact with a grabbable. The player and the grabbable object both had a rigidbody. When the player interacts with the grabbable, the grabbable is, as previously, moved to the top of the player's head, rotated to its Quaternion Identity, and a Hinge Joint is created on both the player and the grabbable object. It allows the object to have a more physics-based movement when grabbed by the player, answering to gravity and movement like jumps more smoothly.

[3] *Portal*, Valve, 2007

This version of the grabbing mechanic was a huge improvement. The movement was smoother and felt real. Grabbing and moving objects around the scene had become intuitive and brought a lot of fun to the game.

B. THROWING OBJECTS

After obtaining the first ability, grabbing, the players in the second level gain the ability to throw objects when they grab them. The throwing is a powerful upward impulse that can be activated by a player grabbing an object or another player. Its implementation, as built on the grabbing mechanic, was pretty straightforward; ungrabbing and applying on the object a force of impulse type.

The throwing is used, for example, to throw a player on an upper shelf, an upper part of the level, to continue the progression of the level.

C. OTHER MECHANISMS

Other mechanisms have been implemented to improve gameplay. One of them is a button system. The player can jump on a button, which activates another mechanism while the player sits on the button.

One mechanism that could be activated by the button was a jumping pad. The jumping pad, when activated, worked like a spring or a trampoline when you jumped on it. It was another great element of fun.

Finally, we also had the idea of a swinging mechanic that we had to give up on due to time constrain. This mechanism would have allowed players to swing on a rope over gaps for example. We managed to prototype swinging using a collection of hinge joints simulating a balancing rope. We never implemented it in the game, but it has potential for future levels.

VI. NETWORKING: MULTIPLAYER AND LOCAL COOP

Having a multiplayer game was the main constrain of the competition. The multiplayer had to be cooperative and online.

Our idea of two bok-choys facilitated the design of the COOP. We just had to implement it. We chose to support two types of networking. The first one was a local mode, where two players could play on one computer, one screen, by plugging in two controllers. The second mode was an online one: with two computers connected by network.

The first multiplayer we established was a local one, as it was the most straightforward. We attributed number IDs to both players to differentiate their controls and attributes. We were then able to plug two controllers in the same build and have a local COOP ready. This has been the most used multiplayer mode for its simplicity. We didn't need two computers but only two controllers, we could test the game quickly. We then improved the local mode with a player manager to help us differentiate the two players and let them communicate between each other.

We established the online networking using Photon, a Unity Extension. The networking worked with a room system. Each player could create a room and host a game. Their room would be then listed in the lists of rooms where his friend could find and join him. This system would obviously cause a lot of issues at a bigger scale. First the server, being local, has to be able to handle the networking. It is also causing a security issue: without any passwords or a private room system anyone could enter anyone's room. Finally, with a large room list the user could be lost without any filter. Our UI was not able to handle a large list too. These

issues didn't concern us much as we were for now working on a much smaller scale, with the only big affluence at the day of the competition [4]. Photon was thus the perfect tool for us, considering the scale of our game and the time scope of the competition.

The Photon Multiplayer brought a lot of issues in the production of our game. The lack of communication between the team working on online and the one working on local multiplayer made networking really difficult. In addition, we were very new to networking and had zero experience with developing a multiplayer game. Our first error was to start with a local multiplayer game. The comfort of the local mode didn't really motivate us to figure out the online mode. We started developing the game as we always did, without thinking much about networking at first. Thus, we did not anticipated issues like prefabs instantiating, control mapping or at-launch initialization. Synchronizing and instantiating the two players was the most challenging part of networking.

[4] Cancelled due to COVID-19 pandemic.

VII. THE ARTIFICIAL INTELLIGENCE

Another constrain of the Ubisoft Game Lab was the development of an Artificial Intelligence in the game.

We chose this AI to work against the progression of the players as an enemy entity. The AI is the mad scientist that generated the two bok-choys. The scientist will walk around the laboratory randomly; if he sees the bok-choys trying to escape he will catch them.

The Artificial Intelligence is a State Machine representing the scientist behaviors (Figure 1). The State Machine has 3 states: Wander, Investigate and Attack. The base state of the scientist is the "Wander" state. When the scientist is wandering, he is not actively looking for the bok-choys but walks randomly in the lab background. However, the players can trigger the scientist: transitioning him into the "Investigate" state. Triggering the scientist could be by making noise by dropping an object, or by simply standing inactive at a fixed position. When the scientist is triggered it goes in the "Investigate" state. He moves to the trigger position and scans the zone with his eyes, representing a moving field of view. If the players enter the field of view, they collide with the field of view: the scientist sees them and enter the "Attack" state. The Attack state triggers the respawning of the players. The scientist goes back in a "Wander" state. In the "Investigate" state, if the players manage to hide or escape though, the scientist will give up after investigating for a certain time and will go back into a "Wander" state.

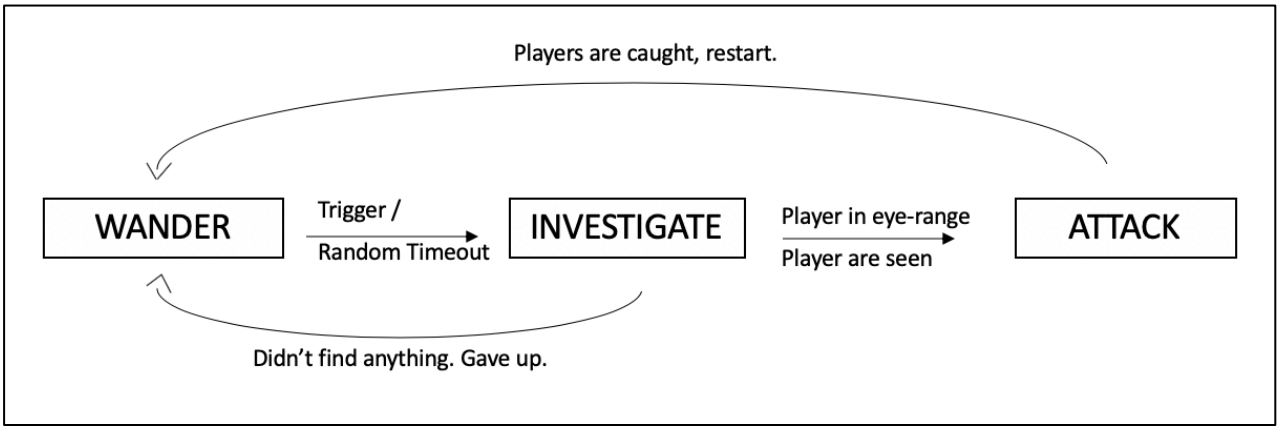


Figure 1: Diagram of the State Machine of the Scientist Artificial Intelligence

The movement of the scientist in the "Wander" state is determined randomly in the lab environment: the background of the level. The random movement is computed by a Nav Mesh Agent in Unity.

This mechanic adds a stealth aspect of the game, adding tension and suspense. The players not only have to solve the puzzles to escape the lab but also have to think about the scientist and hide from him.

VIII. RESULTS AND CONCLUSION

At the date this report is written, we finished the game with 2 levels. The first one is focused on the grabbing mechanic and the second one on throwing. 3 levels were planned at first, but we narrowed down the scope to two levels in order to have longer and more polished levels.

The prototype stayed at a nearly finished state, stopping 2 weeks before the original Ubisoft deadline. The COVID-19 pandemic forced us to pause working on *Get Me Out*. These two weeks would have mostly been work to polish the game; test it and changing the level design to make it more entertaining. We also had to correct a few bugs and test it to make the puzzles finishable. The sound and other triggers for the AI also stayed incomplete. Finally, we were missing the final implementation of the online multiplayer.

The game is still playable and has a lot of potential for a larger scale. We can easily imagine adding more levels and more abilities.

This production of Get Me Out showed us the importance of playtesting while making the game. Throughout the competition we tested the game ourselves, but also invited other students to test *Get Me Out*. This was key to make design decisions according to their feedback; adjusting the controls and modifying the level design.

This game was a huge opportunity to gain experience on a big project. We learned a lot, from coding to organization, team building and communication. We are very proud of *Get Me Out* and we will continue working on this game.

APPENDIX

A. REFERENCES

Unity and Game Development resources:

- Brackeys, YouTube, 2013-2020
- "Better Jumping in 4 lines of code", Board to Bits Game, YouTube, 2017

Games and Inspirations:

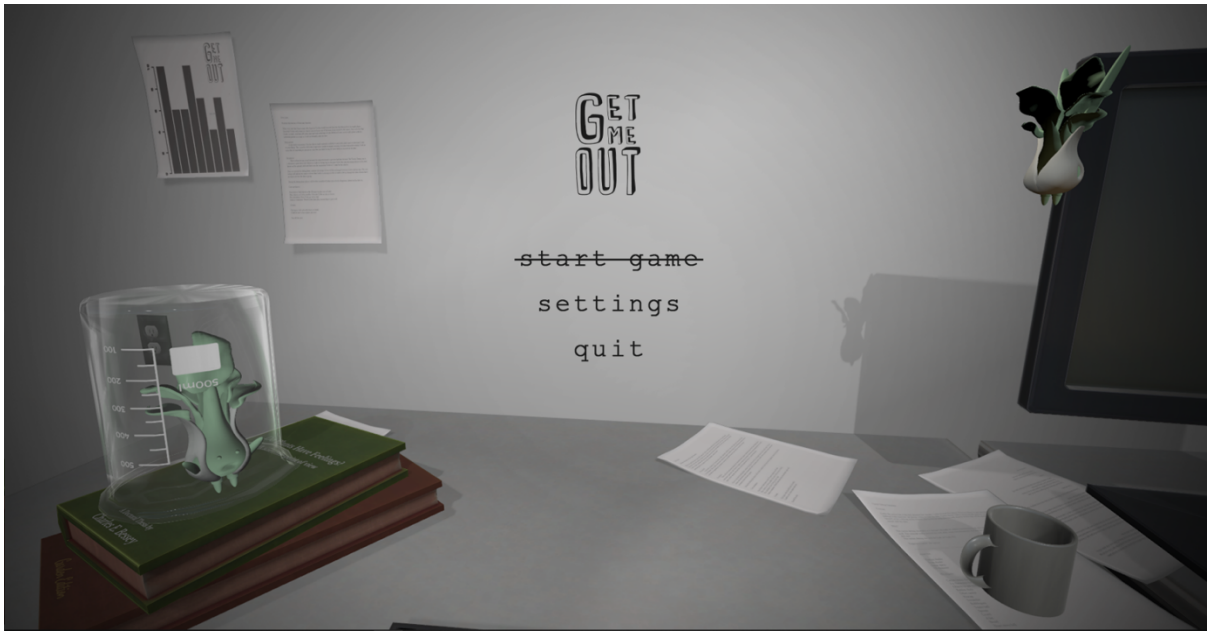
- Portal, Valve, 2007
- Rayman Origins, Ubisoft, 2011
- Rayman Legends, Ubisoft, 2013
- Little Big Planet, Media Molecule, Sony, 2008

Other references:

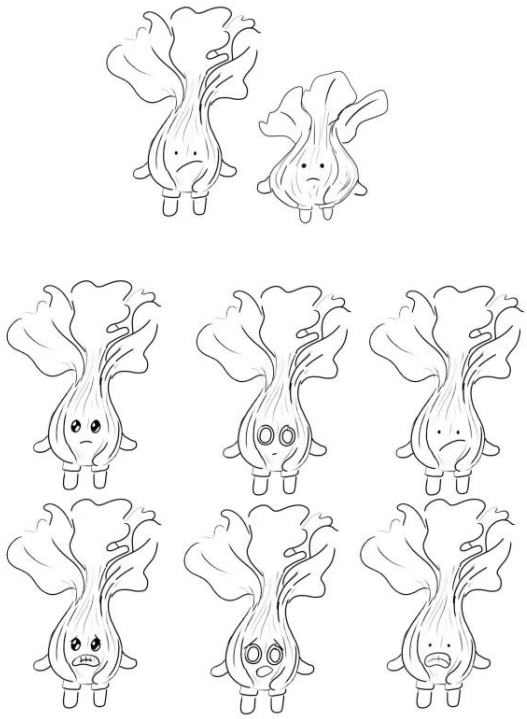
- GameDev McGill, McGill association, for resources, technical help, material and logistics.
- Ubisoft Montreal

B. SCREENSHOTS OF THE GAME

Official presentation image of GetMeOut.



Screenshot of the main menu of GetMeOut.



First drawings of the two bok-choys, main characters of Get Me Out.

Overview of the Level 1 of GetMeOut, March 2020.

Overview of the Level 2 of GetMeOut, March 2020.

C. OTHER GAME IDEAS

a. Time Squeeze

Overview
Players control the parents in a family with young-ish kids. They try to survive day-to-day life and overcome the "time squeeze". Overcooked type gameplay.

Core Loop
The parents will have to cooperate to do all the tasks that present themselves. They will have to complete the tasks before the kids become too impatient, otherwise the kids will start smashing stuff and screaming. (Clean up messes, cook food, read bedtime stories?...). (I feel like the tasks they have to do is not very important to have down right away, we can add some simple ones to begin with and then add more when we iterate).

COOP/Social Mechanics
Players should communicate to do all the tasks in time. Technically they could just not communicate and do what they think is needed (so it would work as an online game without us needing to implement some type of in-game communication system), but obviously it is better to communicate and work together.

Progression System/Loop
Each level is one day (from getting up in the morning and getting the kids to school, until the kids are in bed sleeping). The levels are separated by eg a year of time, so the kids would get older (presenting new problems such as difficult homework, moody teenagers…) and to make the levels harder they can get more kids.

AI Elements
Kids are "enemy" AI if the parents don't do stuff fast enough.
Maybe also in the harder levels they can have a pet or two that randomly smashes stuff, and in the easier levels/in an easy mode they can have a "roomba" to help them clean.

b. Disaster Survival

Overview
Two players work together as a builder and a resource gatherer to generate structures to survive natural disasters.

Core Loop
1. The two players work together to build a structure.
2. A natural disaster occurs and the players work together to repair their structure and survive.
3. The players are rewarded for surviving and keep repairing and building onto their original structure, or they didn't survive and start over.

COOP/Social Mechanics

During the building state, the builder must communicate what resources for the other player to gather and bring to them. During the disaster state, the resource gatherer must put out fires and clear debris so the builder can repair the damage to the structure.

Progression System/Loop
Each subsequent disaster is more intense than the last. As resources are used up, it becomes harder to collect new resources after each disaster. After each disaster, the builder learns new primitive shapes to build and new resources to use, and the resource collector gains the ability to collect different resources.

AI Elements
AI natural disasters. One idea is AI resources that "avoid" being collected.

c. Evolution

Overview
Players control two white blood cells that defend against viruses through "phagocytosis." The players can combine and control a single cell to beat larger enemies and to grow.

Core Loop
1. Players work together to defend against viruses.
2. Players eliminate all the viruses and move onto the next level.

COOP/Social Mechanics
Players have to communicate when to split and combine since some viruses can only be eliminated when split and some only when they're together. They also have to communicate when to come together in order to grow. When they are together they have to communicate on where they're going since they both control one cell. If the two players move in separate directions they split again.

Progression System/Loop
Each level is a new part of the body that needs to be defended, with new viruses.

AI Elements
The viruses are AI. One idea is to also have AI white blood cells that fight alongside the players.