VISCOUS BISCUIT

Game Design Document version 0.2

David Jacobsson
djacob@kth.se

Platon Woxler
platon@kth.se

Konrad Wihl
wihl@kth.se

Matilda Richardsson
mrich@kth.se

Yuxuan Huang
yuxuanh@kth.se

GAME DESCRIPTION

GAME OVERVIEW

Viscous Biscuit is an explorative puzzle platformer based on changing the state of matter of the player character. You move through the world as a little cookie dough blob with the very special ability of changing your own state of matter at will. As you progress, you are presented with different obstacles, which can be overcome by utilizing the properties of the different states of matter with skillful timing. Perhaps there's a thin crack in the floor which can be squeezed through by becoming liquid. A glass door in the way? Become solid as a rock and smash it. If the goal is perched up high, become gas and float like the wind.

STORY AND CHARACTER

Biscuit is an experimental cookie dough concocted in a secret mysterious experimental bakery. They wake up, seemingly spawned from two containers which have fallen from their respective conveyor belts - one containing biohazardous sludge and the other cookie dough. Exploring their way upwards through the maze of strange lab facilities, they gradually discover the ability to change their own physical state and thus overcome increasingly challenging obstacles. However, it turns out there are antagonists lurking in the background, trying to keep Biscuit from reaching the light of day. After each level in the bakery, Biscuit faces off with the mysterious Master Baker, who becomes more and more determined to capture them and cook them in the oven...

GAMEPLAY AND MECHANICS

MOVEMENT

Pretty straight forward - Biscuit can move laterally and jump. Besides these basic abilities, they can also use their special ability which is toggling between different physical states of matter. In the other matter states, they lose the ability to jump, but instead gain other abilities....

CHANGING STATE OF MATTER

With the press of a button, biscuit can change from a (somewhat) solid piece of dough into liquid form, gaseous form, and potentially more forms. This can be done to tackle obstacles which regular biscuit cannot get past, for example running through a thin passage in liquid form. When becoming liquid, gas etc, biscuit cannot be controlled quite as easily - so in order to move more freely they must return to their regular form.

LIQUID STATE

In liquid form, Biscuit cannot jump, but they can assume the shape of their surroundings, just like liquid.

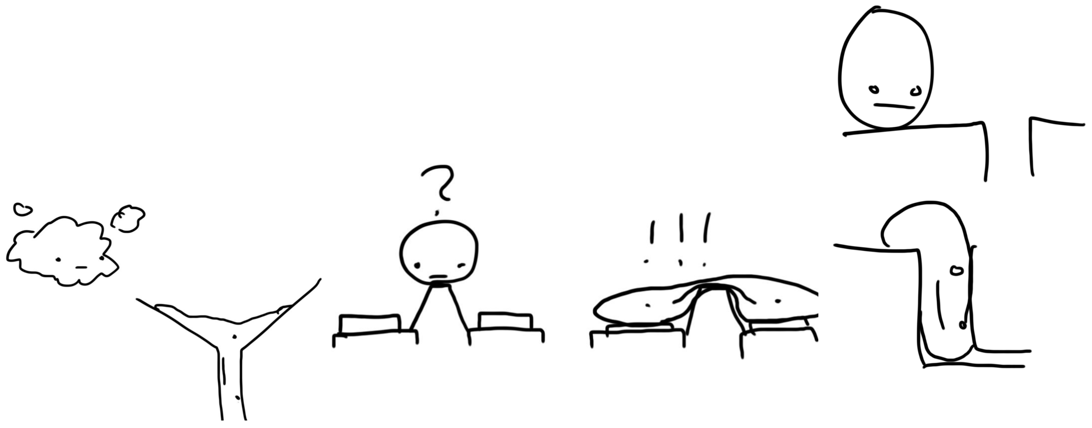


Figure 1. Concept sketches of the liquid state

GASEOUS STATE

In gaseous form, Biscuit can float upwards and through narrow spaces, but is easily affected by the wind

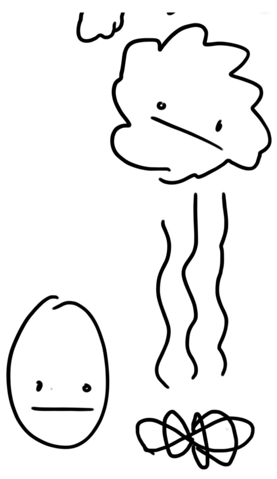


Figure 2. Concept sketches of the gaseous state

MIXED STATE

This is Biscuit's default behavior - they are neither solid nor liquid, just kind of squishy. However, they can move around pretty easily.

SOLID STATE

In solid form, biscuit becomes completely frozen in whichever shape they were previously in. This might be used to get through gates or breakable objects in the environment. This mechanic is not available in the first stages of the demo, but is discovered in the level *cooldown quarters*!

FIRST LEVEL - DEMO

The Viscous Biscuit demo consists of the first “tutorial” level, namely **Baker’s Pit.** This is done deliberately, as we wish to emphasize the physics features and draw the player in. A more ‘vertical slice’ approach would probably throw testers off due to the complexity of the puzzles. Another reason for having the demo be a single cohesive level is to really drive home the tone and atmosphere, which is tricky to do if the player is constantly thrown into radically different environments.
The demo level lays down the format which the rest of the game will largely follow; the goal is to travel upwards vertically, but in order to do so one must first explore left and right to find clues and tools necessary to move further up the bakery. In the following section the different rooms of the demo level will be laid out in greater detail.

SPAWN ROOM

This is the center room, which is also the start screen for the demo. We exclude UI for saving a game, creating a new game etc - not because it won’t be in the final game, but because the spawn room will be the first thing showing after starting a new game.
The spawn room initially contains the viscous biscuit logo, but no player, as well as a "press any key" prompt. Upon pressing, the logo fades and Biscuit spawns.
Besides being a spawning room, the room serves as the central hub between rooms branching off to the sides. This is thought to be a recurring level design pattern in the final game, as the player moves upwards to progress. The room contains the following props/clues:

- A sciency chalkboard, conveniently displaying a clue to press arrow keys + shift to change matter states
- A ceiling fan with a suspicious passage above it, which is turned on and prevents Biscuit from floating up and past it
- A fuse box with lit lamps and two cables leading left and right in the directions of the puzzle rooms, and connected to the ceiling fan

PUZZLE ROOM 1: CHEMISTRY LAB

To the left of the spawn room is a puzzle room dedicated to showcasing the liquid mechanic. The player enters and is ushered through a pressure lock chamber, which prevents the player from using the gas mechanic at all (high pressure, no gas). Inside the room is a collection of glass pipes, which the player must squeeze through in the correct sequence in order to reach the far side and deactivate the generator.

PUZZLE ROOM 2: OVEN BOILER ROOM

This area has less puzzle elements, and serves more to enhance the somewhat dark and mysterious tone of the game, while still showing off the gas mechanic and some more environmental obstacles. The player enters via the right side of the spawn room, and makes their way through a dim, narrow corridor. At the end of the corridor is a burner, which doesn’t deal damage but instead forces Biscuit to enter the gas state. The player then floats up through a ventilation shaft and enters some type of maintenance quarters. By toggling buttons which turn off the burners and open doors, the player may advance to the top of the room and switch off the generator.

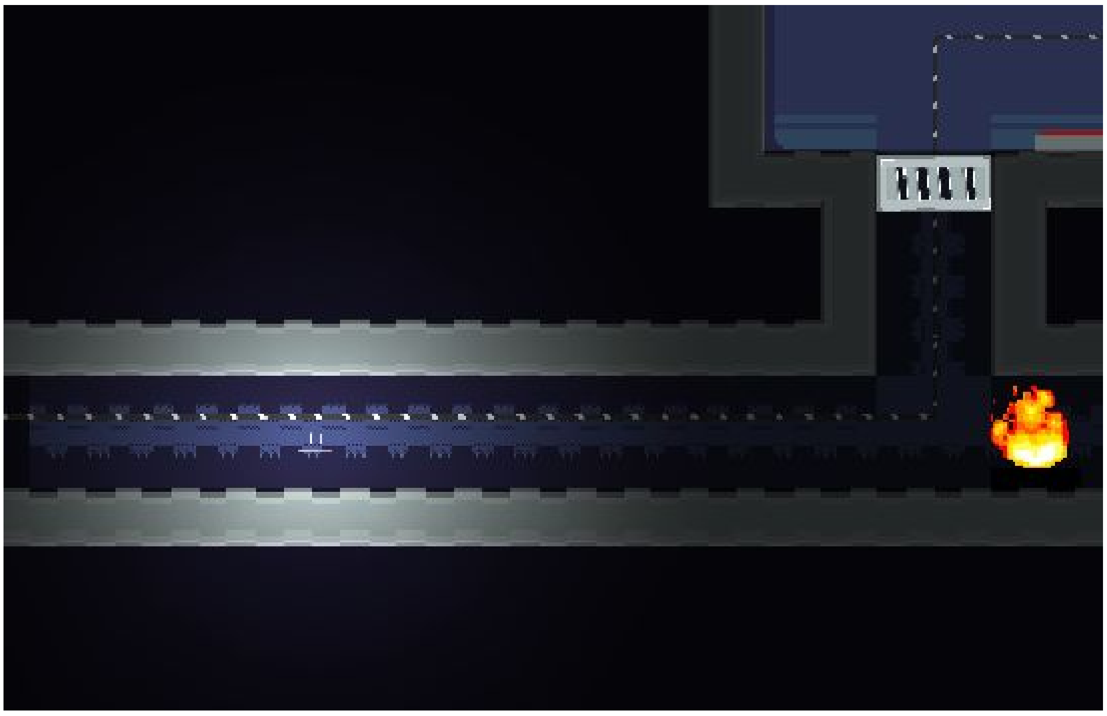


Figure 3. The oven boiler room entrance

ASCENSION ROOM

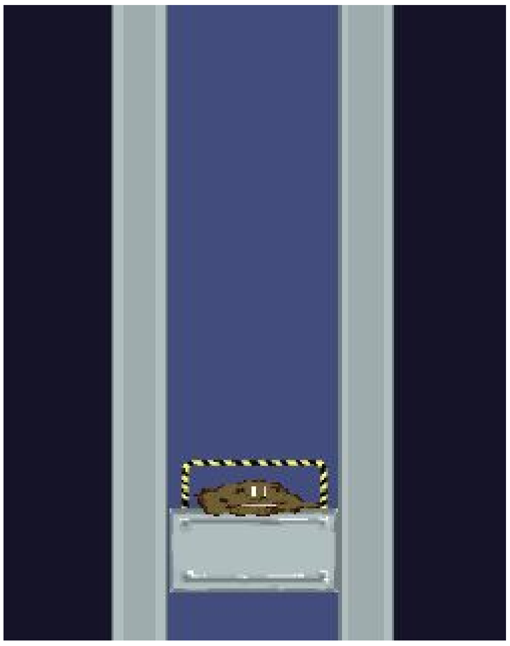


Figure 4. Ascension room elevator shaft

After turning off both of the generators, the player can use the gas state to float up through the spawn room past the now stopped ceiling fan. After exiting through a passage in the ceiling, the final room of the demo is shown. This room is merely for aesthetics and symbolizes the transition from one environment into another. It's simply an elevator platform taking Biscuit up through a shaft, and into the unknown.

OTHER LEVELS

Cooldow Quarters

This level is a storage environment with freezer rooms whose cold environment makes the floor slippery and prevents biscuit from their gaseous state. Ripe with slippery platforming elements and crate pushing puzzles, this area offers a more challenging, slightly less experimental experience compared to Baker's Pit.

Cargo Bay

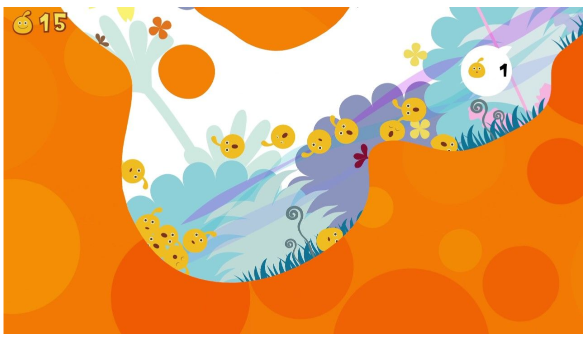
The cargo bay is the first level where daylight can be seen. The glass pipes from the chemistry lab in Baker's pit return but this time as pneumatic transport systems for little containers of ingredients/messages etc which are brought into the bakery from the outside. Naturally, the player can sneak into these pipes and follow the flow of the production chain from within. The player also has to dodge automatic delivery vehicles, but may get the opportunity to control them as well.

Control tower

Following the message delivery pipes in the cargo bay, the player suddenly gets routed to the human populated upper parts of the bakery. The player is confronted with scanning surveillance cameras, security doors, and most importantly lab workers in yellow hazmat suits who seek to impede Biscuit's progress by sending them down trap doors and barricading office spaces. The environment is a mix of dry office landscapes, sterile lab environments and gloomy, dusty maintenance shafts. On the top of the control tower is the final face off with the nefarious mad baker scientist, who will attempt to escape their creation (Biscuit) via a helicopter pad. Why? Biscuit just wants answers!

LOOK & FEEL

Some influences for the general tone and aesthetics of the game are Love You to Bits and LocoRoco:



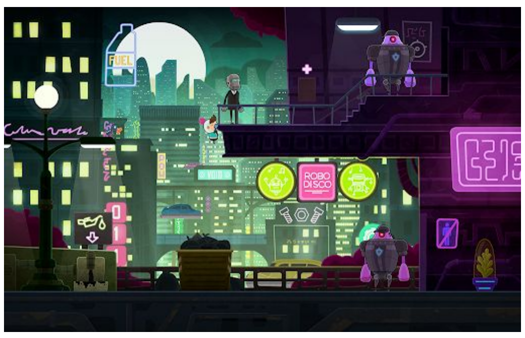


Figure 5. Left: LocoRoco. Right: Love You to Bits

Similarly to these games, the graphics are to be colorful, playful and heavily stylized. If possible, special rendering tools will be utilized to achieve a slightly grainier, more retro look although this might be difficult due to the real-time physics having a higher granularity.

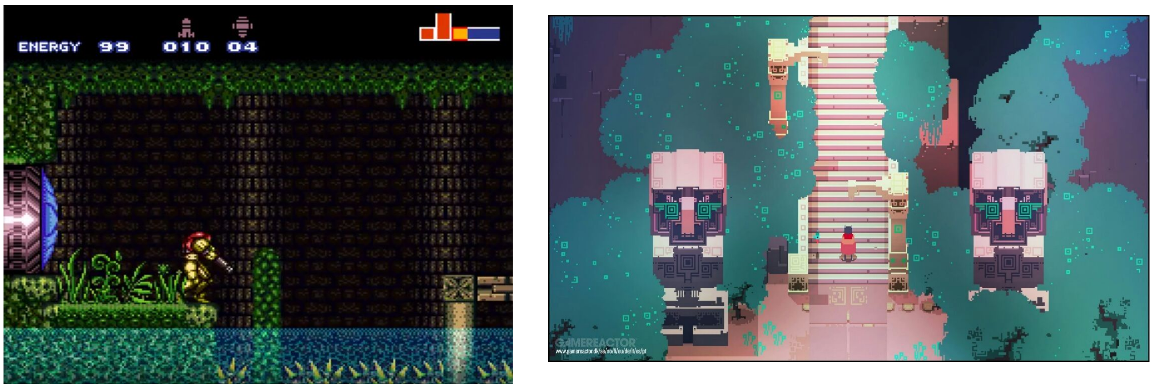


Figure 6: Retro looking games; one actual retro example (Left: Super Metroid) and a more modern one (Right: Hyper Light Drifter)

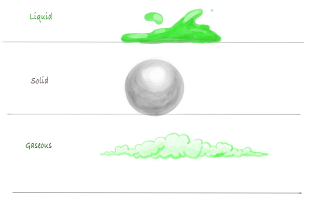
Feel-wise, Viscous Biscuit will be medium paced; an even blend of puzzles and platforming elements. The idea is that the puzzles will require timing and switching between the different matter states in the right order to reach the goal of the current room.

Figure 7. Puzzle/physics gimmick-esque games with different pace - Left: VVVVV. Right: Braid

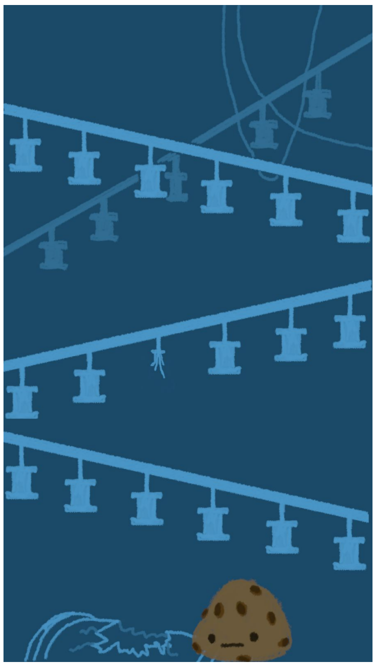
DESIGN HISTORY

This section will store some historical notes, images and changes made during the course of development.

WEEK 45-46

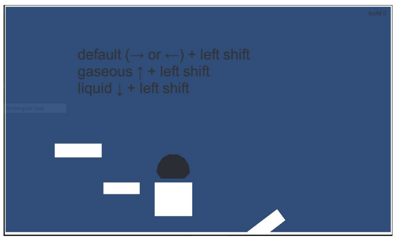


The initial game concept: A two dimensional platformer with heavy emphasis on the puzzles and physics. Not much was decided in terms of design, other than the indifferent face of the main character. Another difference from the final product is that there wasn't any clear direction in terms of overarching narrative, storytelling or pacing. The sketches suggest more of a context-free, playful tone - the type of game you would play for five minutes on your phone while waiting for the bus, and then forget.

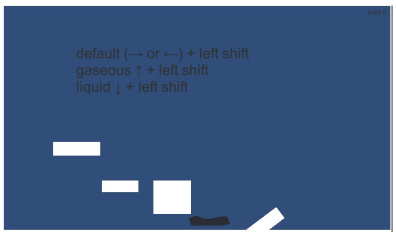


WEEK 47

Here there were some early examples of gameplay and exploration of different aesthetic choices. The look and feel of graphics and gameplay were very much in the testing stage, but the different test maps contained different styles which would later converge. As far as the gameplay goes, the physics component was somewhat dominant, but mainly due to the physics programming still needing fine tuning. At this stage the player would need to be very deliberate in order to squeeze through certain passages, and the movement was quite slow. This would change in the following weeks.



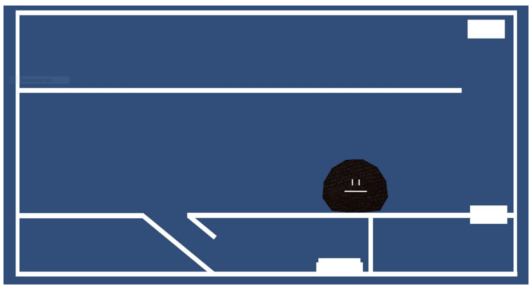
default (→ or ←) + left shift
gaseous ↑ + left shift
liquid ↓ + left shift

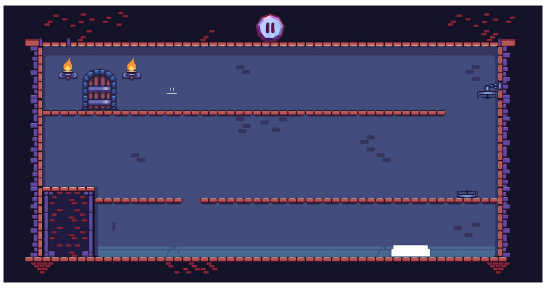


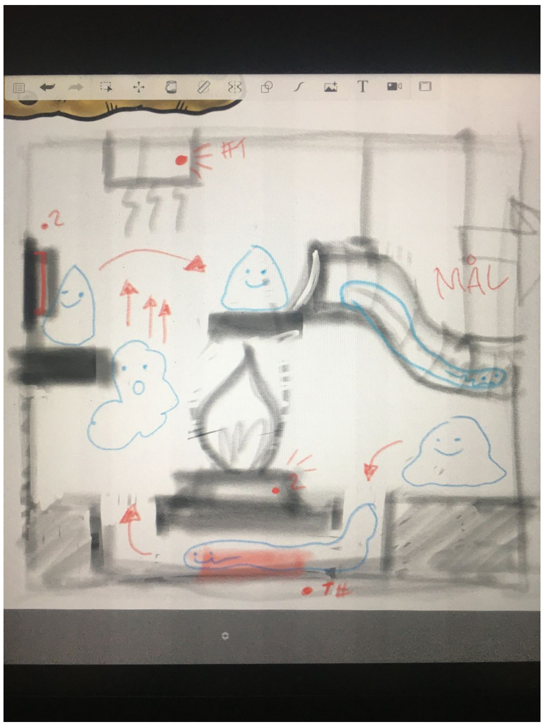
default (→ or ←) + left shift
gaseous ↑ + left shift
liquid ↓ + left shift

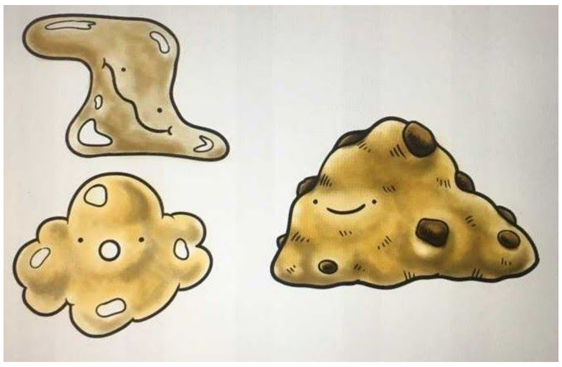
WEEK 49

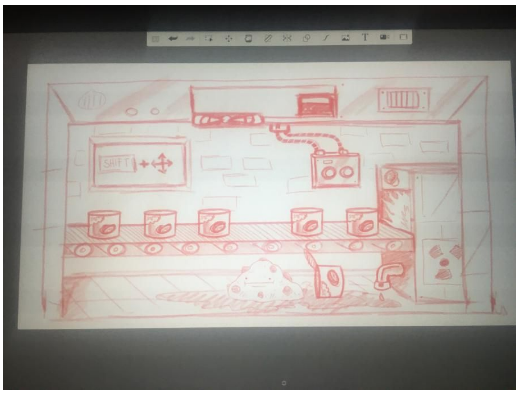
This is where the playable levels with working physics started to merge together to get an actual feel for the final product. As can be seen in the rightmost figure, there were experiments made with open assets to establish a tone. Even though this example hints more towards a medieval or fantasy type game, it's clear that the general aesthetic has become darker and more convergent. There were also improvements made regarding the control of the blob; faster movement allowed for much more of a conventional platformer feel, which is what we ultimately strived for.

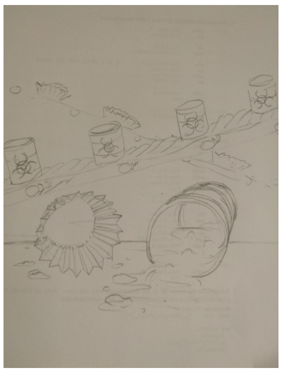





#1
2
MÅL
#




SAFT



TECHNICAL BIBLE

This section describes all the important parts about how to organize the project codebase as well as practical teamwork.

CODE-BASE STRUCTURE

We used a standard model view controller setup for this project. The persistent state of the game was stored in a controller object holding things as current scene and game progression. The blob parameters or "settings" are fetched from a scriptable object at initializing and the blob is separated from its controller.

As for file structure we follow standard unity praxis with our scripts, prefabs and Resources stored separately.

For general code formatting we used Capital letters at the start of method and function names and lower case letters for variable names. As a team we mostly used camel case as is the praxis for unity projects. This goes in line with the prescribed C# best practices.

TECHNICAL IMPLEMENTATIONS

BLOB SETUP

The player object or the "Blob" as it is known, consists of a set of joint objects. The blob object acts as a center joint which works as a parent to all the other joints. A joint is connected to other joints in various ways, which we will go over later. The user sends input data into the controller which keeps track of the blob state and movement parameters and forwards the correct data into the blob object and makes the player move.

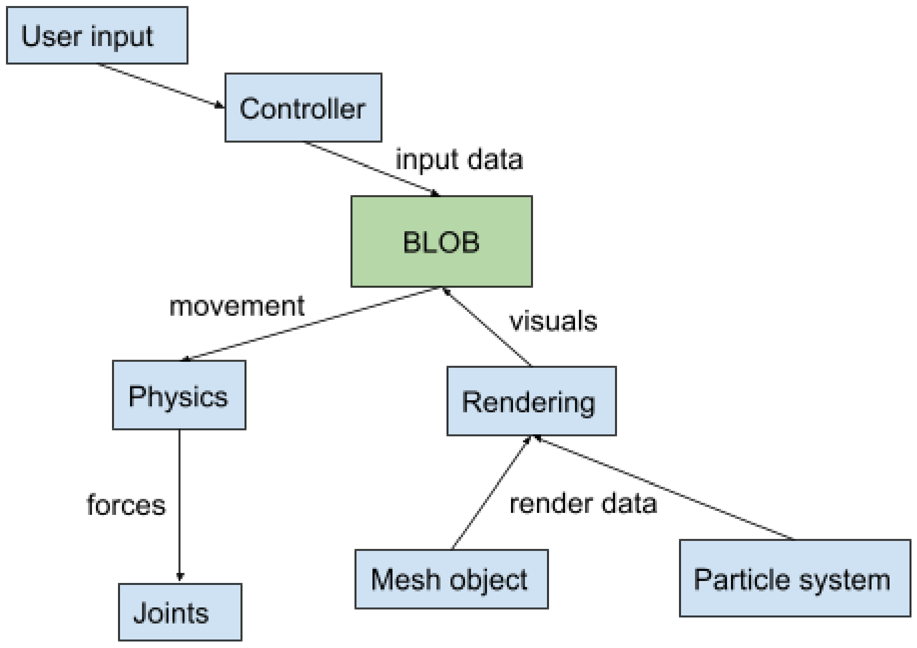


Figure 8: diagram describing overview of blob structure and data flow

The blob object also holds the mesh object responsible for displaying the mesh and texture of the player. The particle system is used for creating and displaying the gas state of the blob and is also a child of the main blob object. See fig blobHierarchy for a visual explanation.

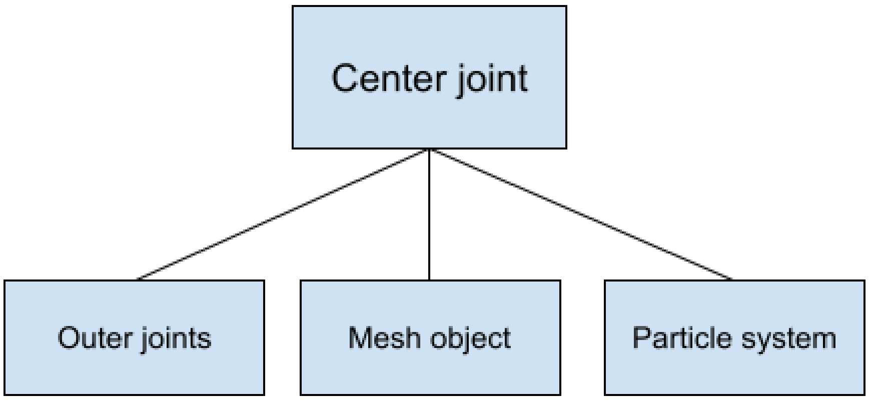


Figure 9. Diagram describing hierarchy of the blob

JOINT OBJECT

A joint object itself holds several other objects used to control the feel and behaviour of the entire blob. Each joint object's own circle collider is as big as to always be in contact with its neighbours collider. The spring and distance joints are connected with other joint objects in a way that creates the soft body physics we were after. See figure 10 to see the relation of the blob and its components.

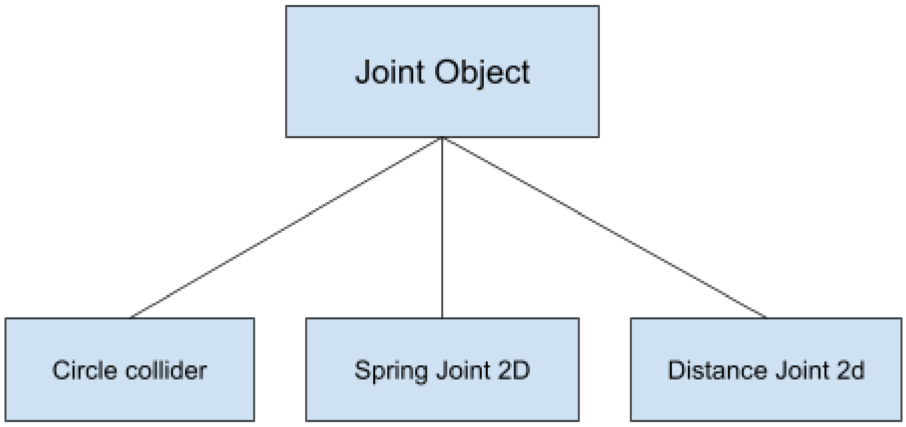


Figure 10. Diagram describing hierarchy of a joint object

BLOB STATE PHYSICS

Most of the blob physics are altered by changing the relation and parameters of the joints that make up the blob, see figure 11 details. Every joint of the outer joints is connected to its immediate neighbour by something called a distance joint (red). This distance joint keeps a constant distance between the joint and its neighbour. Each joint is also connected to the neighbour one over and the center by something called a spring joint (green). This acts, as the name would suggest, as a spring and gives the blob its bouncieness properties.

By adjusting the stiffness of the spring joints connecting the different joints we can achieve different characteristics. Less stiff springs will give us a more liquid feel while a stiffer blob will give us something more akin to a rubber ball.

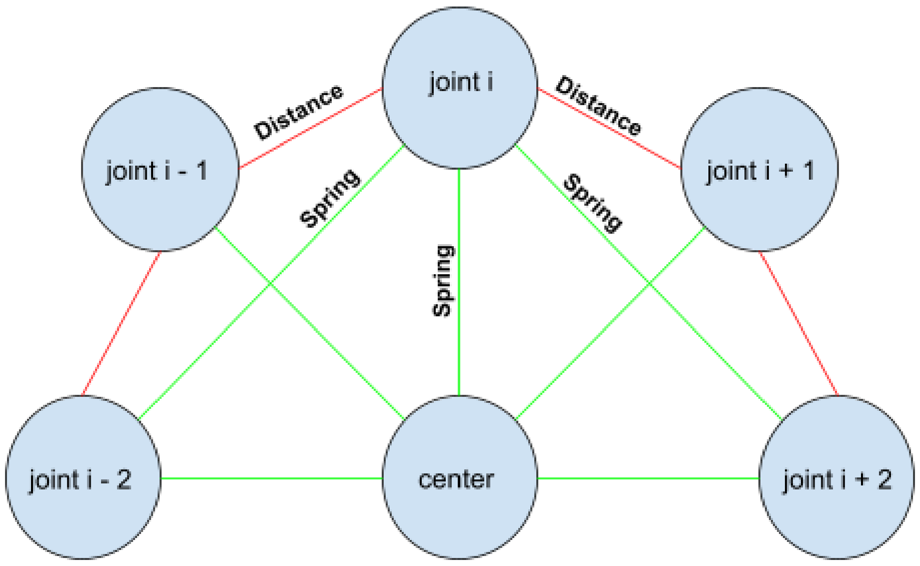


Figure 11: diagram describing relations between joint objects.

To improve stability and controllability we also added some upper limit distance joints from each outer joint to the center joint. This joint works as the regular distance joint but only activates on an upper limit, you could imagine it to behave as a stiff string. We also set an angle restriction between the outer joints to stop the blob from compressing too much under pressure. This constriction could also be used to give the blob a completely solid feel if we restrict the movement enough.

MOVEMENT

In this section, the movement components of the Blob are described. Each three main components described are presented in more detail below. See figure 12 for a broad visual representation of the components.

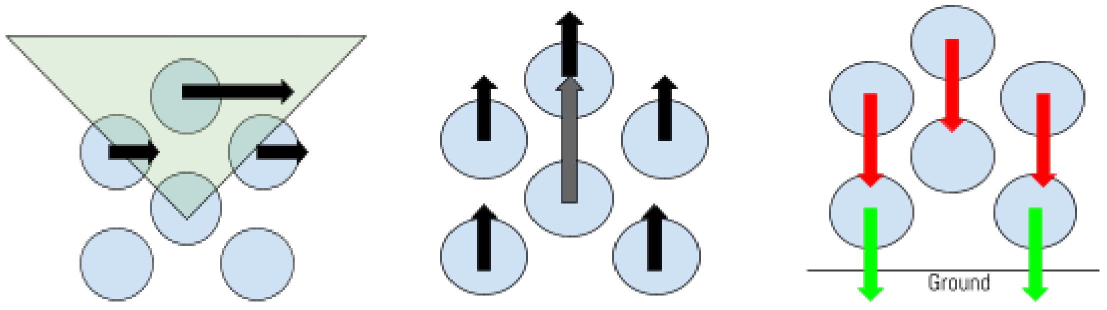


Figure 12: describing horizontal (left) and vertical (middle) movement and ground check (right)

HORIZONTAL

In order to get the feel we were after we wanted to achieve some kind of rolling motion when controlling the blob. This is achieved by applying a force to every outer joint proportional to its angle to the center joint. This makes the upper joints more affected by the force and creates the motion we're after. The force is also applied to the center blob. See figure 12 for detail.

JUMPING

When jumping we apply an upwards force on the center joint and we also apply a portion of that same force to every outer joint so the center joint is not pulling all the weight. See figure 12 for detail. When the player is gas form, a constant upwards force is applied in the same way as previously described.

GROUND CHECK

Before the player is able to jump, we do a ground check and see if the blob is grounded. This was achieved by using raycasts, originated in the center of every joint in the blob, going straight down. If any of the raycasts hit the ground, the blob is grounded. To let the raycasts know what is ground, we set every ground or platform to the layer Platforms. This is illustrated in the figure to the right, where the raycasts are illustrated as arrows. Green arrows indicate that the raycast has hit the ground, and red arrows indicate that they have not. See figure 12 for detail.

GAME PROGRESS

The progress of the game is saved in the Game State Controller, which is a game object used in all scenes. The Game State Controller is created as soon as the game starts, and is set to never destroy after it is created.

In the Game State Controller, the position of the blob is updated every time a player moves into a new room. The position in the previous room is saved and used in the next room, where the blob will start at the same position as it was last in the previous room.

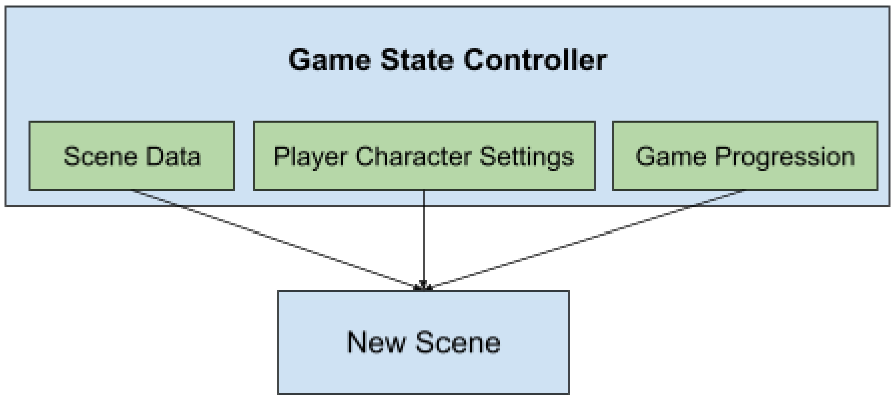


Figure 13: Structure of the game state controller object

The current game progress is also saved in the Game State Controller, such as the current scene, the previous scene, and currently finished levels. We keep track of the scenes by using integers, where the integers on the scenes are decided through the build of the game. The level progress is saved by letting every level have a boolean, with the start value set to false. When the boolean is set to true the level is finished. The boolean for each level is set by the generators, where the booleans are set to true when the generators are turned off.

2D LIGHT IMPLEMENTATION

For 2D lights, Unity's 2D light system Lightweight Render Pipeline v.7.3.1 was used. All props and materials in the game were thereafter upgraded to go through the pipeline. Due to this, some blob materials got a lighter tone, which was fixed by darkening all blob materials in the inspector. To add more realism to the game, we also created a script making it seem like the lights had a soft flicker. This was achieved by adding a script that randomizes the intensity on a light, between a maximum and minimum light intensity given in the inspector. In addition to this, we also added a script that changes the color on the fusebox lights, from green to red when a level is finished, to further add realism.

ANIMATIONS

The animations used in the game were all stop motion animations. Every frame in the animations are separate sprites, made in artistic computer software such as Krita and GIMP. The animations were thereafter created automatically by Unity by dragging them all into a scene in the Unity project. To get the animations to suit our style, we modified and slowed down the speed for each animation in Unity's animation controller.

In addition to this, we also created scripts for when the animations should be turned on and off. This was for example done for the generators, where we in the Start function check if the user has already finished the level, and in the Update check for a trigger when the player will press a specific button in the scene, to turn off the animation. Moreover, a script for turning off the fan animation was made. Here we check in the

Game State Controller, if the player has finished both levels, to then turn off the animation. The animations in both scripts were simply turned off by disabling the animator component for the game object.

TESTING

Since we will be working in an agile workflow with more short time goals rather than fewer long term goals we will implement smaller player testing after each major goal or sprint is complete. This lets us immediately see if we're moving in the right direction and can carry on, or if we should abandon the current path and find something new. The player tests will thus be short and mostly focus on the newly implemented features. Towards the mid-end stretch of the project we will conduct some more in depth testing to get an overview of the state of our project.

DEVELOPMENT PROCEDURES AND STANDARDS

In this section the agile workflow and policy is described as used by the project team. It considers the practical development of the game itself, as well as the version control procedure used.

AGILE WORKFLOW

The general concept is a custom designed derivative of the kanban and scrum process making use of *milestones* and *tasks*. With the project and team being on such a small scale, fully utilizing existing Scrum frameworks include much more organizational overhead than is necessary. From previous experience within the team, it was determined that making use of Instead, a simplified version was devised that prioritized centralizing in one single place all the current priorities, goals and tasks that need completion in one single place. A general overview of the agile workflow for this project is described below in figure 14.

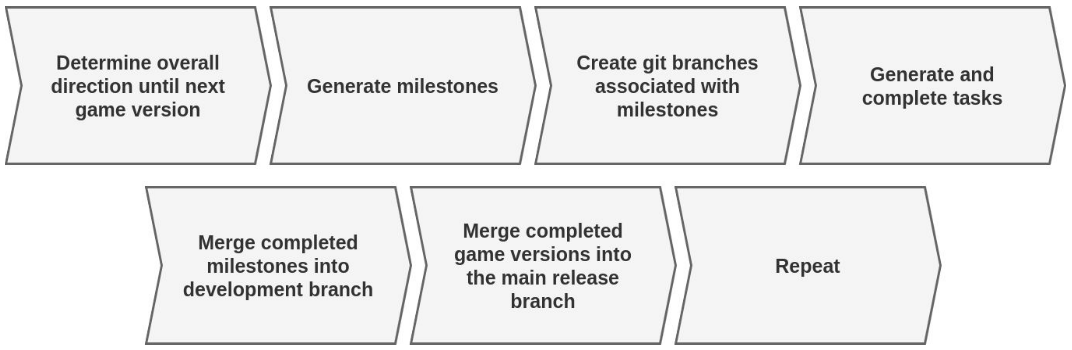

Determine overall direction until next game version
Generate milestones
Create git branches associated with milestones
Generate and complete tasks
Merge completed milestones into development branch
Merge completed game versions into the main release branch
Repeat

Figure 14: Diagram describing the agile workflow of the project team

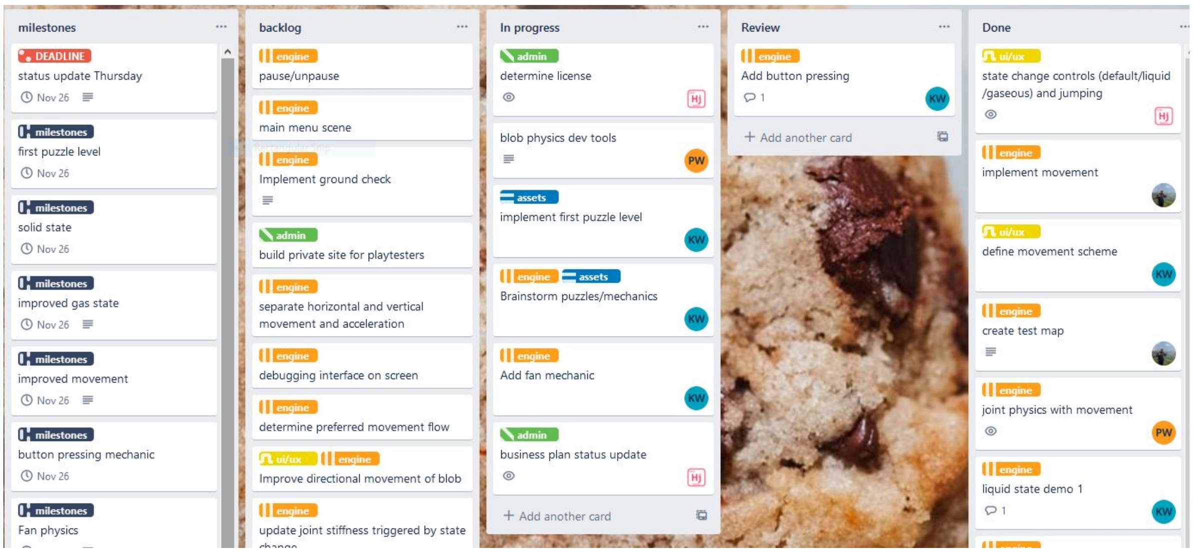
This agile workflow is made up of three main components; *Goals, Milestones,* and *Tasks.* An abbreviated description of each component is described below in table 1. The general goals are determined through discussions and brainstorming sessions with the whole team. It is then the art directors job to lock down a specific target vision which the project will be striving towards. This vision describes essentially what the game will be like, what broad components it will consist of and which criteria needs to be fulfilled for these goals to be achieved. Once the goals are set, the team derives specific milestones to work towards.

The milestone is intended to be an intermediate step between direct, practical tasks and the general goals. They can be in any resolution that is deemed appropriate by the team. Essentially, the level of detail a milestone contains depends on several factors, such as level of project progress in total, what the goals are and how well they are defined. The important aspect to remember for the team when deriving milestones is that they represent one step towards one goal and not in any sense the final product. As soons as the milestones for any given goal are completed, new goals will be formulated. In the event that the milestones do not complete the goal as intended, new milestones will simply be added, or the goal redefined based on the experience gained from the milestones completed so far. This way, the team is constantly working in accordance with the most well defined and granular idea available at any given time.

Table 1. Description of the three agile workflow components.

Concept	Description
General Goals	Current high level targets describing the current direction of the game. The goals exist in the context of the greater vision of the game, but must be a self contained step in the development process, e.g a game demo or technical presentation.
Milestones	Distinct steps towards achieving a goal. These should strive towards differentiating the different aspects of the goal and represent mutually separate steps.
Tasks	High resolution component, representing a very finite and concrete set of actions. Examples are for instance a particular improvement to the jumping mechanic, or adding a first draft of a new level.

Milestones themselves make up a number of individual tasks. A task needs to be a smaller, fairly self explanatory, step that achieves a clear part of the milestone. The tasks are defined in real time by the team members. It is also important that both milestones and tasks are actively prioritized internally.



milestones
- DEADLINE — status update Thursday — Nov 26
- milestones — first puzzle level — Nov 26
- milestones — solid state — Nov 26
- milestones — improved gas state — Nov 26
- milestones — improved movement — Nov 26
- milestones — button pressing mechanic — Nov 26
- milestones — Fan physics

backlog
- engine — pause/unpause
- engine — main menu scene
- engine — Implement ground check
- admin — build private site for playtesters
- engine — separate horizontal and vertical movement and acceleration
- engine — debugging interface on screen
- engine — determine preferred movement flow
- ui/ux, engine — Improve directional movement of blob
- engine — update joint stiffness triggered by state

In progress
- admin — determine license — HJ
- blob physics dev tools — PW
- assets — implement first puzzle level — KW
- engine, assets — Brainstorm puzzles/mechanics — KW
- engine — Add fan mechanic — KW
- admin — business plan status update — HJ
- + Add another card

Review
- engine — Add button pressing — 1 — KW
- + Add another card

Done
- ui/ux — state change controls (default/liquid /gaseous) and jumping — HJ
- engine — implement movement
- ui/ux — define movement scheme — KW
- engine — create test map
- engine — joint physics with movement — PW
- engine — liquid state demo 1 — 1 — KW

Figure 15. Image showing an example of the customized kanban board used by the project team

Goals are stored in a designated place, for instance a temporary subsection of this GDD. Milestones and tasks meanwhile are gathered in a kanban board tool, in this case Trello. An example of this project's Trello board can be viewed in figure 15.

VERSION CONTROL

Git is used for all version control. There are three main levels of hierarchy, depicted below in figure 16. Primarily, the foundational layer is the release branch, which contains the latest tested and verified version of the game that is currently released to players. Above that is the develop branch, which represents milestones currently being implemented into the game. From here, branches can be created both for individual tasks, as well as more encompassing groups of tasks that pertain to the same milestones. For daily work, branches are used often to implement tasks in an isolated manner. Whenever commits are merged into one of the upper level hierarchy, i.e Release or Develop, a formal pull request is used.

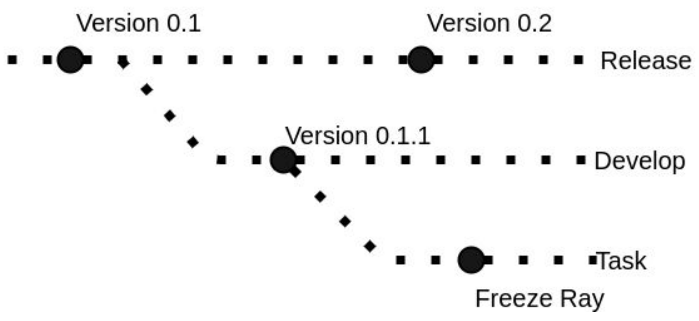


Figure 16. Hierarchical structure of branching

COMMUNICATION CHANNELS

Table 2. Dedicated channels used for internal communication.

Channel name	Description
Active Discussions	Ongoing talks
Brainstorming	Discussing new ideas
Updates	Info flow for the team with only posts containing info the team needs
General Organizing	Repository of descriptions about how to work. Practically used as a quick reference for policies etc.
Storage	Important messages, images and other files that the team wants quick access to is stored in this channel. Examples might be concept art images or other visual materials produced during the course of discussions.

Discord has been determined as the primary tools for communication between the team. Other tools have been used, but the excessive functionality of for instance Slack was found to be unnecessary. Additionally, the video chat functionality of Discord, with persistently hosted video channels, was found to be the most appropriate for a smaller team. On top of that, a number of dedicated text channels are used to organize the ongoing conversation. The different channels and a brief description can be found in table 2.

ART BIBLE

In this section, concept art and assets along with a guideline to the game's artistic vision.

GENERAL AESTHETICS AND GAME-PLAYER COMMUNICATION

The overall style used in the game is pixel art style, and a setting where the player has a side view of the scene. All scenes and levels have either a static camera setting where the player always sees the entire level, or a camera setting where the camera always follows the player, depending on the level design. The overall artistic vision of the game is a kitchen and lab based setting. Overall, the background, grounds and platforms in the game are tile based. Whereas, details and objects, especially interactive objects are sprite based.

Game instructions are solely communicated through visual cues and hints. The players do not get any obvious instructions or hints, but are encouraged to explore the game by themselves, and through this figure out the game mechanics and the goal of the game. The most obvious instructions communicated to the player is a visual asset used in the start screen of the game, showing a subtle hint on game mechanics.

COLOR SCHEME

Table 3. Scheme of main colors used

⬛	⬜	🟨
Dark background #1D212D	Creme white #FBF4EC	Yellow text #F7B543

SPRITE ASSETS

The visual assets were made up of bespoken assets created by the Viscous Biscuit team, and some premade assets were purchased from various artists. See Appendix A1 for a list of a complete list of sprite assets.

ANIMATIONS

All animations were bespoken by the Viscous Biscuit team. The overall style of the animations was a stop motion, retro style. The idea was to create animations where it was still obvious that the animations were made up of only a few frames, to give the animations that retro stop motion style.

Figure 17. The animations used on the generator.

Figure 18a. A reference used when some animations were created. This reference was especially used when we created one of the fan animations.

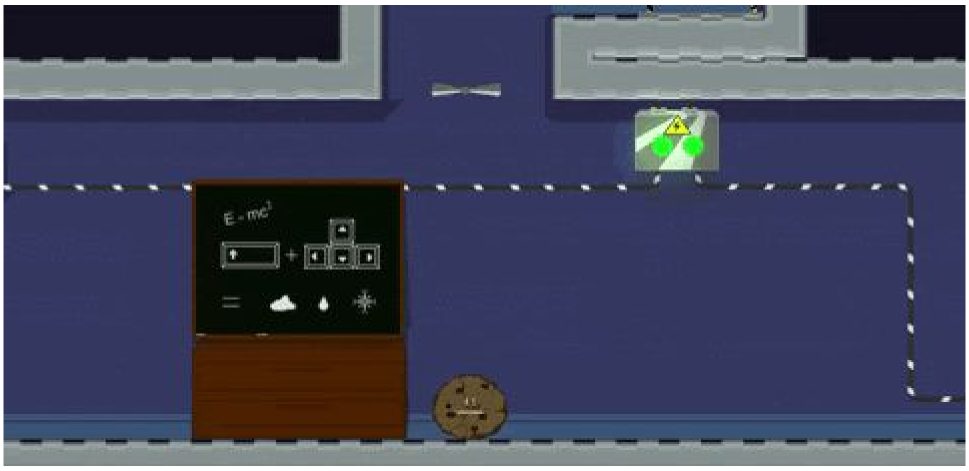


Figure 18b. The animations used in the start screen. The animations are stop motion and you can see the different frames used.

LIGHTS

To create more of an effect and atmosphere in the game, Unity's 2D light system Lightweight Render Pipeline introduced in Unity v.2018.1 was used. In all scenes we added a global light, that lights up all things in the scene somewhat, but we used different intensities in all scenes.

SPAWNING ROOM

In the spawning room we kept a quite high intensity on the global light, since we wanted the user to be able to see everything in the scene and get an idea on how to solve the puzzle. In addition to the global light, we also added lights on both lamps on the fusebox. These lights change color depending on whether a level is finished, as well as have a soft flicker to give the impression of the fusebox getting different amounts of power from the generators/cables.

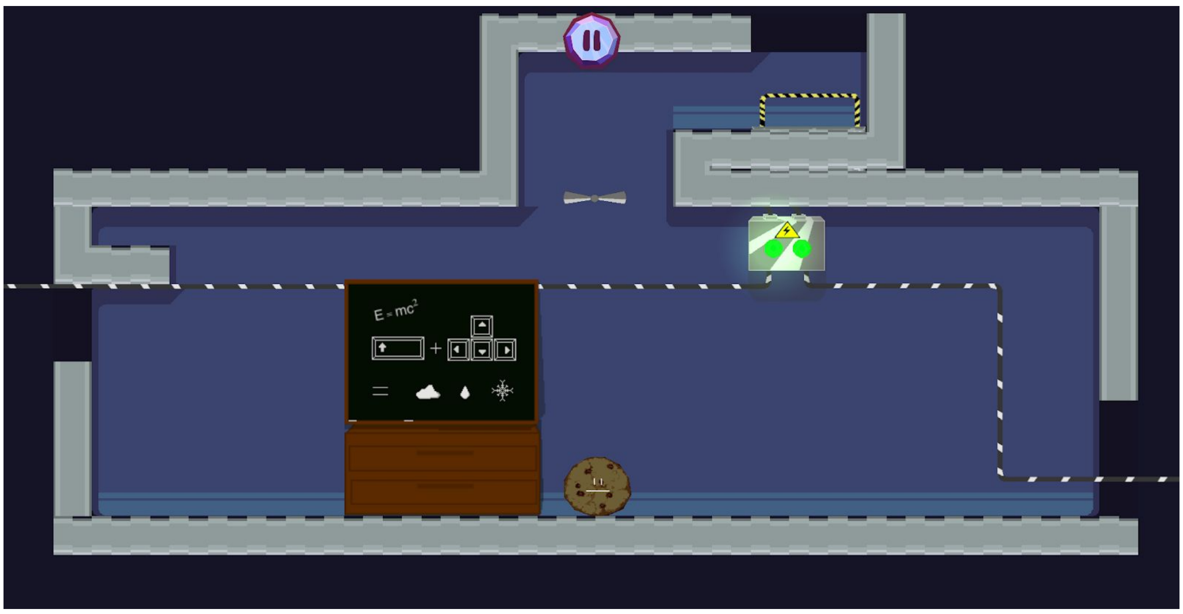


Figure 19. Lights in the Spawning Room.

GAS LEVEL

In the gas level we lowered the intensity on the global light, since we wanted to create an illusion where it seemed like the blob was moving through pipes. To still give the player some visual information on where to go and what to do, we added a light with higher intensity covering every area where buttons existed. In addition to this, we also added a green light with soft flickering to the screen on the generator, to make the generator seem more lifelike. Finally, we added a circular light to the blob that always follows the player, so that the player can see well what is close to the blob.

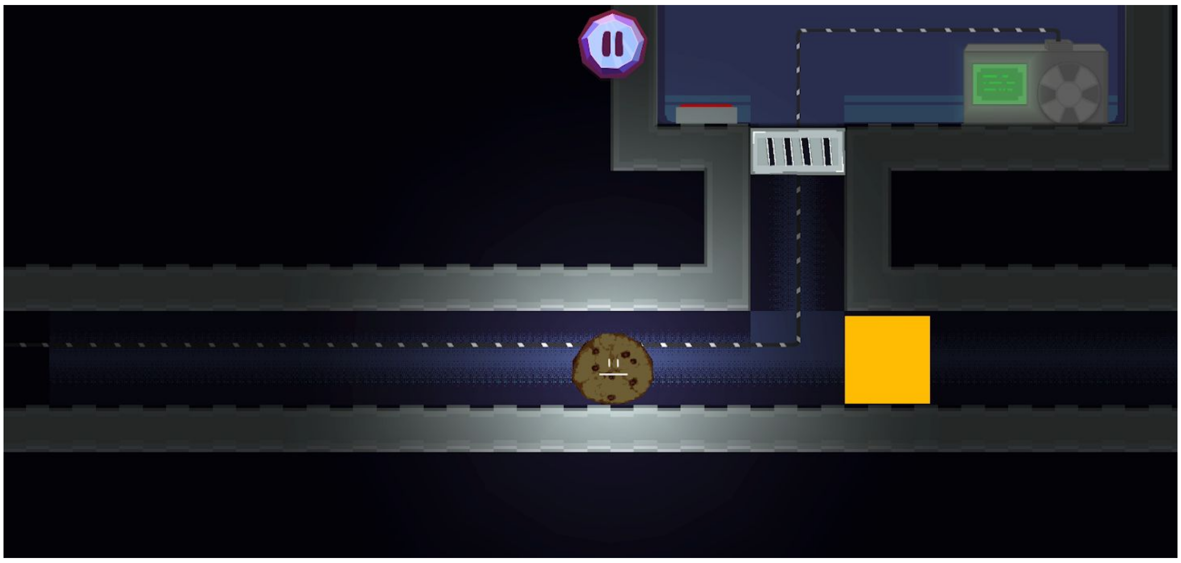


Figure 20. Lights added to the Gas level. In the upper right corner you can see the brighter area with buttons, as well as the light added to the screen on the generator.

LIQUID LEVEL

In the liquid level, we kept a brighter global light, since it is important for the player to see the entire room to be able to solve the puzzle. In addition to the global light we only added a green, soft flickering light to the generator.

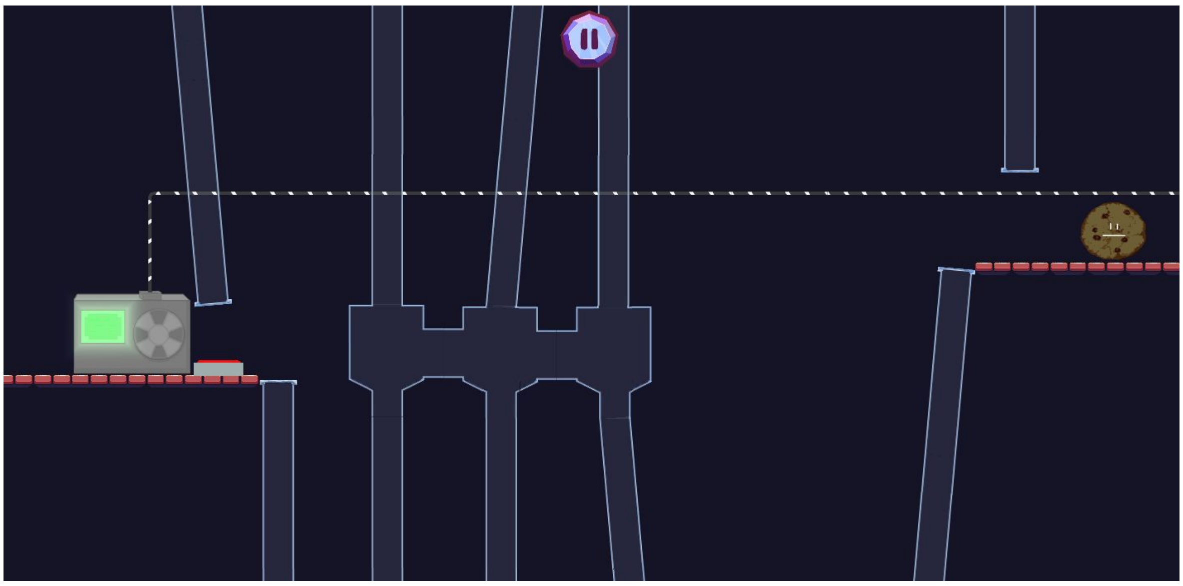


Figure 21 .Lights in the Liquid level.

AUDIO ASSETS

The game uses open source free to use audio assets for effects, and music by an indie game music composer[1]. The music has been recorded using a combination of Bandlab and Logic Pro, and was further edited using audacity. The game contains audio cues for jumping, changing states and environmental factors, besides the soundtrack.

The audio cues were implemented using the standard Unity AudioClip and AudioSource classes for loading and playback.

BUSINESS PLAN

TEAM

This project is realised by a team of 5 people. Each team member will have developer responsibilities primarily, with our particular skill-sets naturally making us more adept at different things. Each team member and their specialization is listed in table 4.

Table 4. Areas of specialization for each team member.

David Jacobsson Team Manager	Konrad Wihl Creative Director	Yuxuan Huang Website, Sound Design	Platon Woxler Physics	Matilda Richardsson Art Assets & Animation

TIMELINE

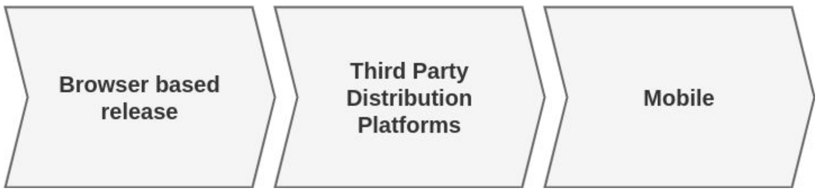


Figure 22. Sequence of platform development over time

Viscous Biscuit is fundamentally a small scale project that aims to be as accessible as possible to its players. As such, the natural choice for primary platform is a browser based game. In the long term however, the game is not limited to only this version. While the browser version is the most appropriate first step,

[1] 'Nosfra', the composer of the Viscous Biscuit main theme https://soundcloud.com/nosfra

there is also room for a release on third party distribution platforms like Steam, GOG, Google Play store etc. Once the game has built up a core following, it will be ported and released on a selection of these platforms. Lastly, the game will also be released on mobile platforms, iOS and Android.

In terms of the game release itself, the team envisions a whole series of subsequent releases that can keep going for at least 3 to 4 years after initial release. First of all, the main adventure release. This is the initial game telling the story of Viscous Biscuit. Shortly of the initial release, there will be a smaller pack of challenge maps that offer more time trial oriented puzzles. For the longer term, there will be a recurring release schedule of holiday and genre based adventures that are of a similar size to the initial game.

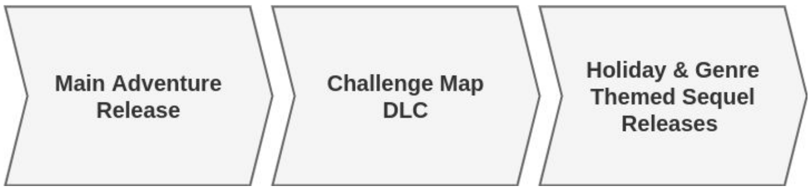
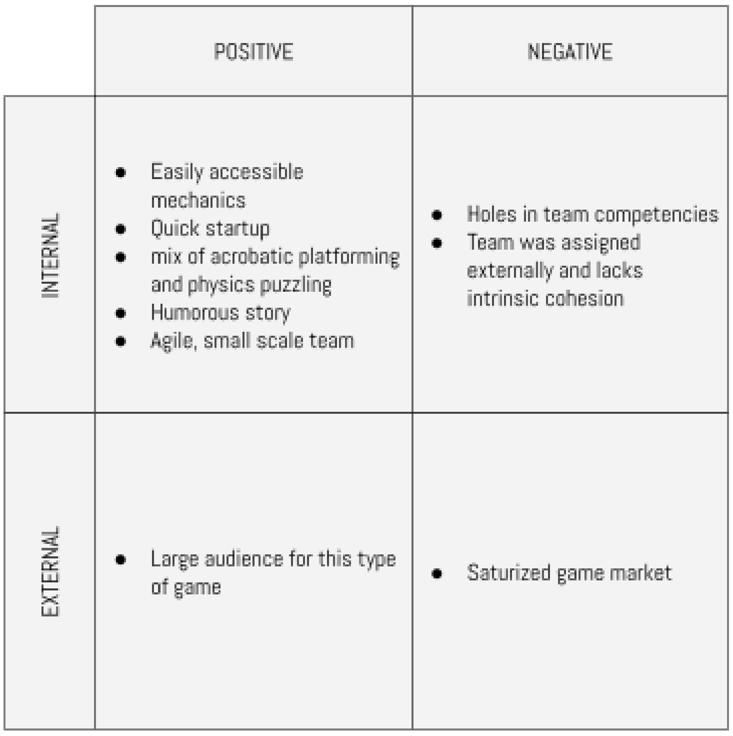


Figure 23. Main release plan

MARKET ANALYSIS

This section is an overview of the marketing aspects for this game. It contains a risk assessment for the team, an analysis of the target audience as well as a marketing strategy.

RISK ANALYSIS



	POSITIVE	NEGATIVE
INTERNAL	• Easily accessible mechanics • Quick startup • mix of acrobatic platforming and physics puzzling • Humorous story • Agile, small scale team	• Holes in team competencies • Team was assigned externally and lacks intrinsic cohesion
EXTERNAL	• Large audience for this type of game	• Saturized game market

Figure 24. SWOT analysis overview

This project has several strengths associated with it. The mechanics lend themselves to being easily accessible, but can also be scaled to several levels of complexity that fits many different styles of difficulty. It is also very easy to get into with the low barrier of entry. The humorous story is another point of uniqueness and strength that speaks in the games favour. However, due to the nature of the context in which the team was brought together, the team has a number of shortcomings that may prove to be substantial obstacles. For one, the team was thrown together based largely on coincidence. Each team member was selected from a pool of students that had no preexisting relationships and there is a risk that the team will never achieve an acceptable level of cohesion and effective cooperation. Overall, creative differences have been handled by following an established hierarchy, but as the team did not choose each other this may not be sufficient. Furthermore, there is a number of indie games of a similar aesthetic that have performed well in the past, and the adventure and puzzle genre make up a substantial portion of the genre distribution as of 2020[2] which might indicate a potential for this game. However, the indie game market is quite saturated, presenting a large challenge sticking out in all the noise. Lastly, the team is not complete in terms of the necessary competencies. While there is some experience in all fields, the team is not especially experienced with game development.

TARGET AUDIENCE

This game intends to be casual and very accessible, while still leaving room for several levels of difficulty. The barrier for starting a game will be low, with quick startup and time from boot-up to game starting will be short. This allows the game to be played by anybody whenever they want to take a break from work, whatever else they might be doing. The expected audience is primarily indie and casual gamers in the 12-40 age range. By being available directly from the browser we will be able to reach the largest segment of our target audience. In the future, ports of the game will be developed for additional platforms (Figure 22). Additionally, the humorous story will be a driving factor that we expect will help speak to our targeted segment.

MARKETING STRATEGY

The intention is to make this game a low effort development. The marketing strategy is envisioned as a circular momentum, with the three main components: *Community Building, Influencer featuring* and *Classical Advertisement:* all leading into each other. Platforms like Youtube and Twitch will serve an important role by having prominent affiliates play the game and create an interest. Some advertisements will be used, but this is not a prioritised tool.

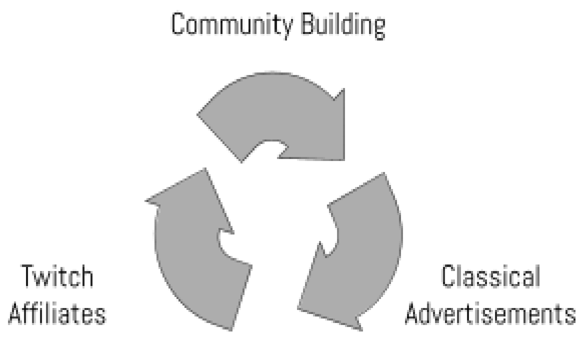


Figure 25. Circular model describing the marketing strategy cycle

[2] According to a ESA breakdown of video games statistics from 2020 url:https://www.theesa.com/esa-research/2020-essential-facts-about-the-video-game-industry/

COMMUNITY BUILDING

By actively interacting with the player base and promoting a living community around the game that rewards active participation by the users, the team can gain effective spread via word of mouth. This community will also be a very valuable source of player feedback and testing.

INFLUENCER PARTNERSHIPS

The most important initial vector for spreading interest in the game and achieving exposure will be gained through being featured by prominent streamers online. This can be accessed both through paid exposure through direct partnerships, and also by leveraging the community and offering free review copies of the game.

ADVERTISEMENTS

Some resources will be set aside to produce and feature classical advertisement on important sites like youtube, twitch, IGN.com etc. This tool will not be relied on especially heavily as most players effectively block advertisements in their browser. There is however some potential for audience reach by releasing an advertisement campaign when the time is right.

BUDGET

The game will be bought once in full and connected to the user account on the website, which in practice will be a google or apple account. Additional releases will constitute content of similar size to the original game and will be purchased as part of a season pass, or single purchases. The approximate cost per player for the game will be 3 to 5 USD. The production will require an initial financing amount of 140K USD. Each cost is described in more detail below and an overview of the first 4 years of operation is detailed in table 6.

Table 6. Overview of the first four years of operation

	Sold Units	Revenue	Salary	Fixed	Variable	Marketing	Total Costs	Total Net
2021	10000	$30,000	$45,000	$500	$900	$23,200	$69,600	-$39,600
2022	25000	$75,000	$45,000	$500	$2,250	$23,875	$71,625	$3,375
2023	50000	$150,000	$45,000	$2,495	$24,750	$36,123	$108,368	$41,633
2024	100000	$300,000	$45,000	$2,495	$49,500	$48,498	$145,493	$154,508

SALARIES

Each team member is estimated to put approximately 30% of a full time workload in this game per year. For calculations, salaries are approximated to about 30K per year.

FIXED COSTS

Costs pertaining to standing tools and resources used by the team. There will not be a need for office space in the initial years, and the revenue will be low enough that software suites like Unity are prized at their lower brackets[3]. The prize is expected to increase within 2 years as the revenue increases.

[3] According to this comparative prize plan for the Unity development suite, url: https://store.unity.com/compare-plans

VARIABLE COSTS

These costs are characterized by scaling directly with use. Web hosting and payment services fall under this category. In the initial years the game will mainly be sold on the website, allowing for cheaper payment options to be utilized, such as Paypal[4]. For coming years, when third party platforms come into the picture which charge a considerably larger portion of the revenue[5], the variable costs will increase substantially.

MARKETING

These costs are very broadly approximated at about 50% of the other expenses for the project. This is a complete guess based on the expectation that marketing will make up a majority of all costs at least for the initial years of operation.

PARTNERSHIPS

A number of partnerships will have to be established in order to set up the infrastructure required to successfully launch and sell the game. First of all, a development environment is needed to work. Several options are available, among which two are Godot or Unity. Both are appropriate for teams with the experience level of this team. The game will require a hosting platform for players to play the game. Amazon offers ready to go cloud solutions for storage and network traffic that allow the team to quickly and easily get the game deployed. On top of the hosting, some manner of store platform needs to run in order for players to buy the game. And lastly, partnerships with social media platforms such as Twitch will be necessary.

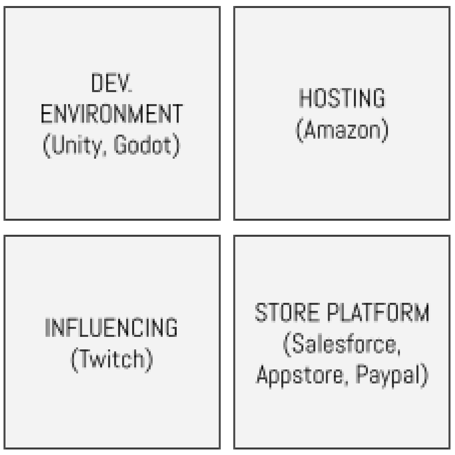

DEV. ENVIRONMENT (Unity, Godot)
HOSTING (Amazon)
INFLUENCING (Twitch)
STORE PLATFORM (Salesforce, Appstore, Paypal)

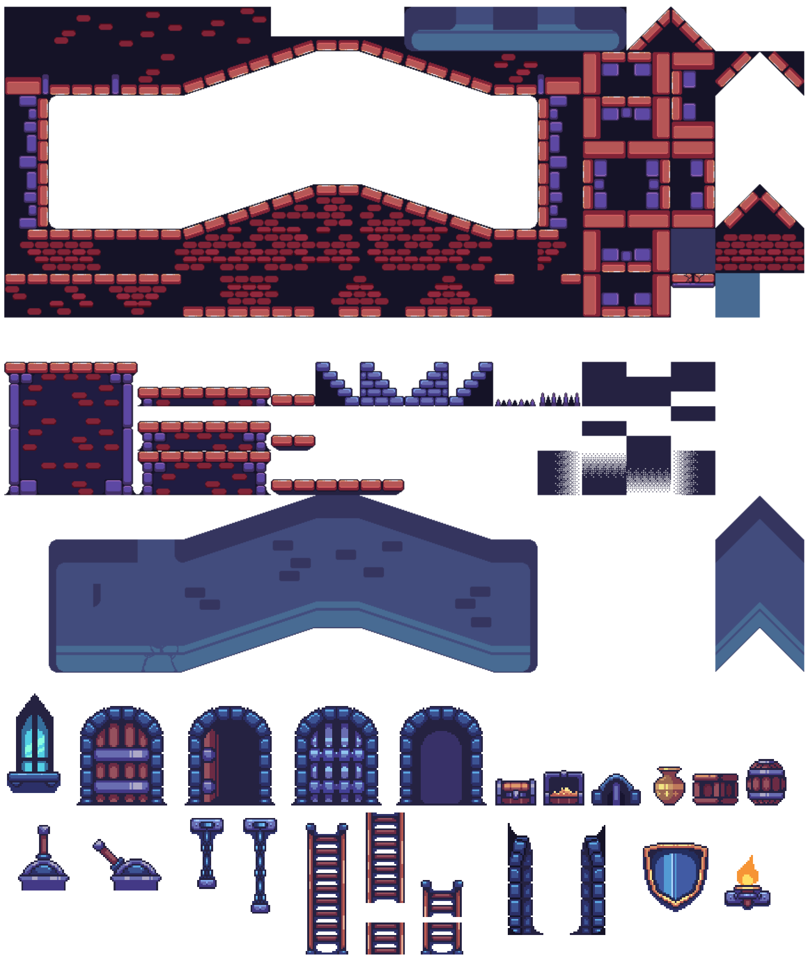
Figure 26. Necessary partnerships for successful introduction to market

[4] Average costs for Paypal payment services https://www.paypal.com/us/webapps/mpp/merchant-fees

[5] Average prizing model for software platforms like Google Play store url:https://www.techrepublic.com/blog/software-engineer/app-store-fees-percentages-and-payouts-what-developers-need-to-know/

APPENDIX

A1. ASSETS



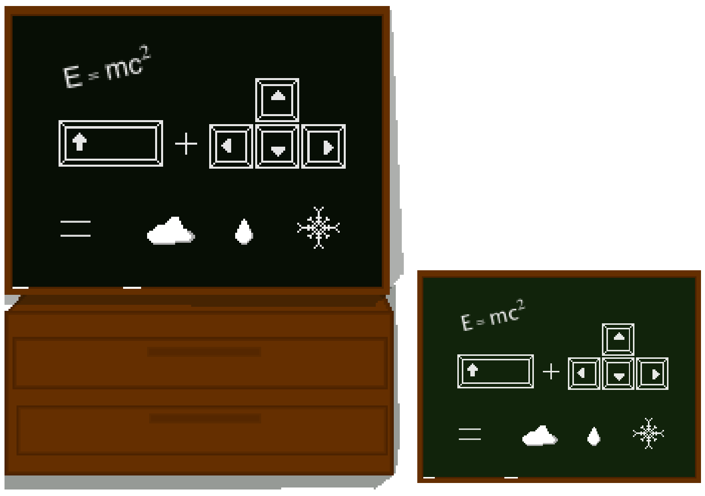
Tile's and props used from the Medieval pixel art asset pack created by Blackspire[6].

[6]Link to the package at Unity's asset store:
https://assetstore.unity.com/packages/2d/environments/medieval-pixel-art-asset-free-130131

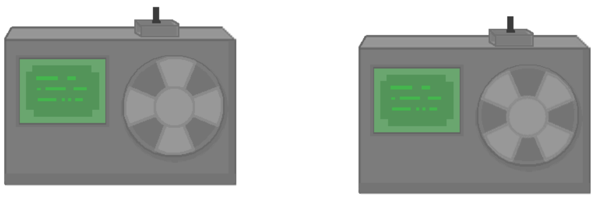
Tile's used from the Roguelike Cave & Dungeons pack created by Kenney Vleugels[7].

Title props used in the main menu scene to tell the player the name of the game.

[7]Link to the creators website: www.kenney.nl

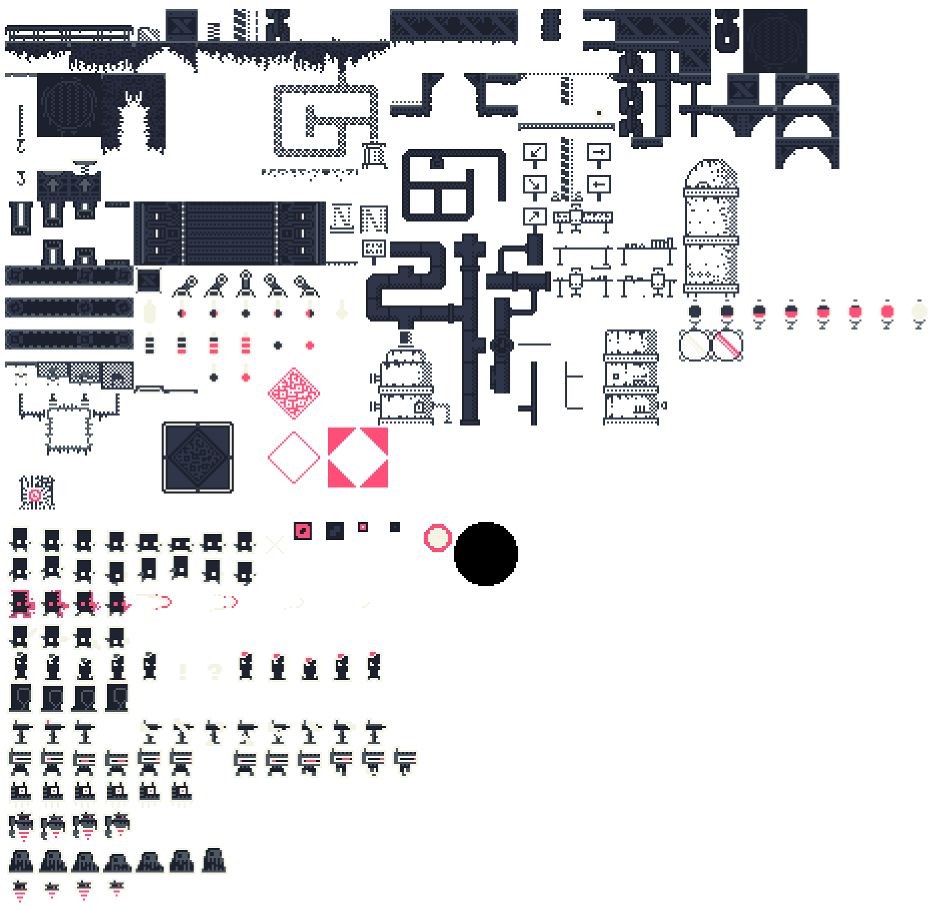


Two chalkboard props used in the first scene of the game to give the player hints on how to change states on the blob.

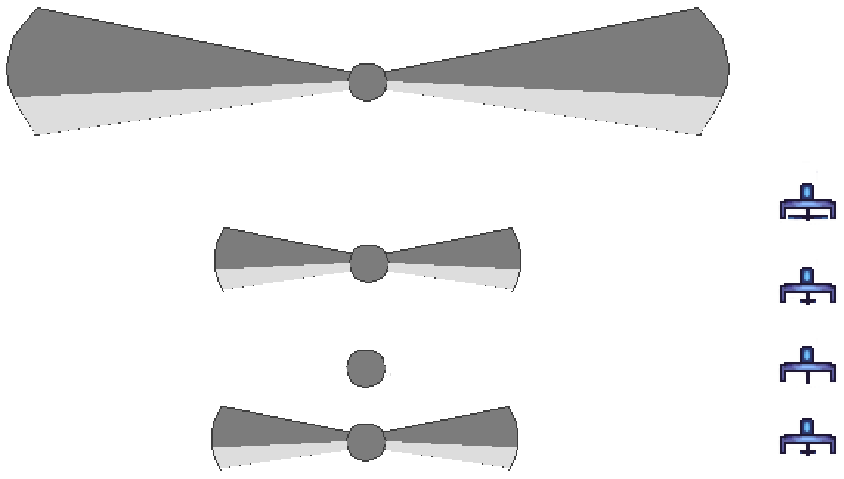


Generator prop, which is animated so it looks like the wheel is spinning. It is used in all current levels as a goal for the player to reach and turn off to finish the level.

Cable props, also animated to look like electricity is going from the generators to the fuse box. It is used to give the players a visual cue to where the generators are, as well as information on if a generator is on or off.

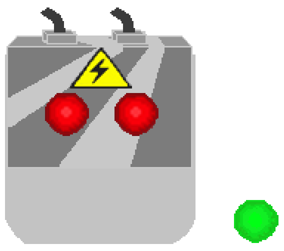


Tile's used from the Dark Industrial pack [8].

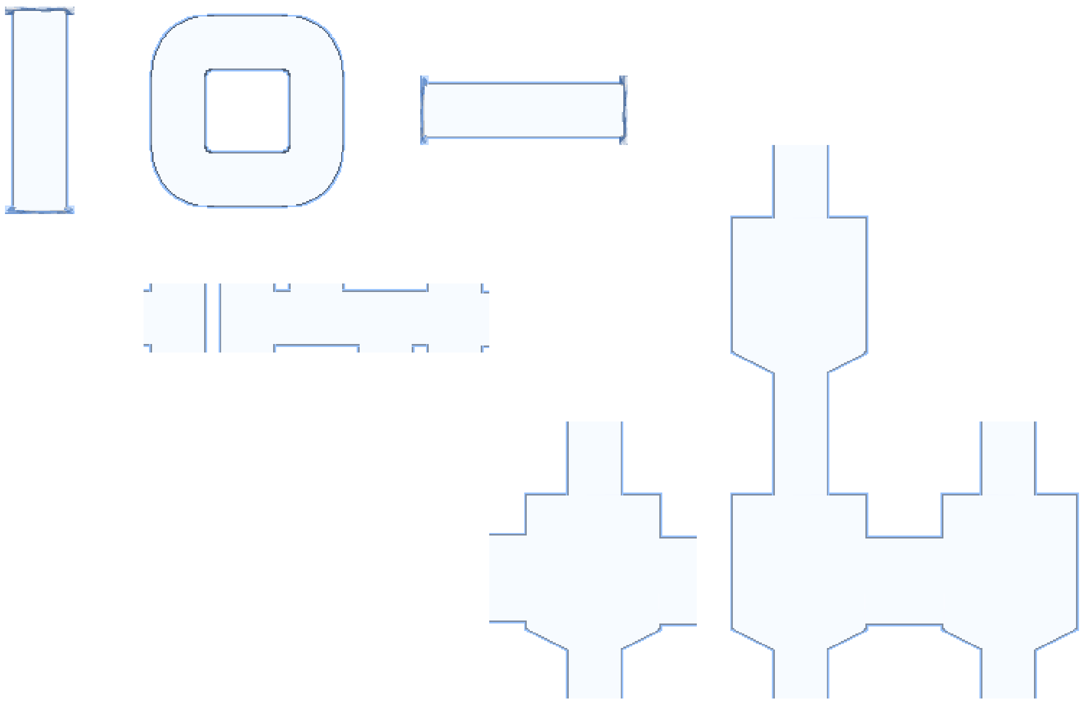


Two fan props, where both are animated. The fan to the left is used in the first scene, to show the player where they cannot go, and that they have to finish the other levels to pass it. The fan to the right was created to be used later as an obstacle in upcoming levels.

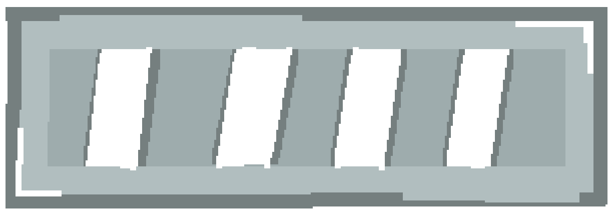
[8]Link to the package at the creators website: https://0x72.itch.io/16x16-industrial-tileset



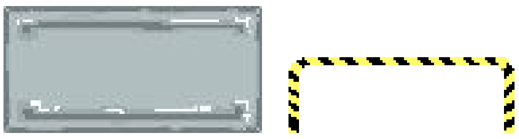
Fusebox and green light, used in the first scene, as a visual guidance to the player on which levels are completed. Red lamps tell the user that a level is finished, and green that they are not finished. The green lamps are placed on top of the fusebox sprite in the game.



Pipe props, which are used as tiles in the game. The pipes are currently being used in the levels to force the blob to be in liquid form, since they are too small for the blob to be in solid form.



Ventilation grate prop. The ventilation grate is supposed to be used as an obstacle for the blob, that the character can only pass in the liquid or gas state.



Elevator prop, used to transport the player to upper levels. The idea is to later use this as an indicator to show that the player is reaching higher levels, and that the levels and the degree of difficulty is greater the higher you go.