Slippery Bash

Game Document

Team Ice Guys

Akbar Suriaganda

Moritz Naser

Omar Ahmed

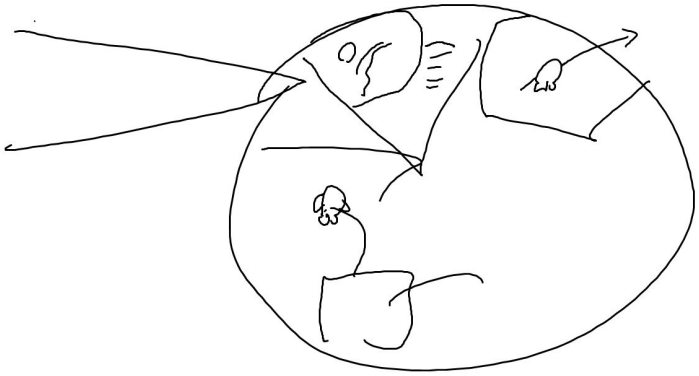
Tarek Elsherif

Game Proposal

Game Description

Game Idea

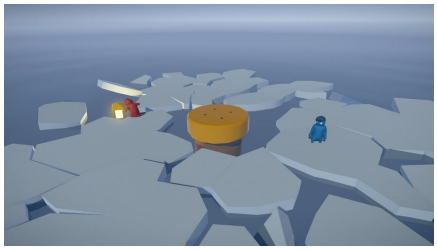
Slippery Bash is a top-down 3D arcade game where players are pitted against each other, using the environment and events to knock other players off the platforms, and become the last Ice Golem standing! The game utilizes real physical properties, inspired by the lab theme, wet and slippery, to allow the players free-for-all struggle through round after round of escalating chaos until one victor remains!

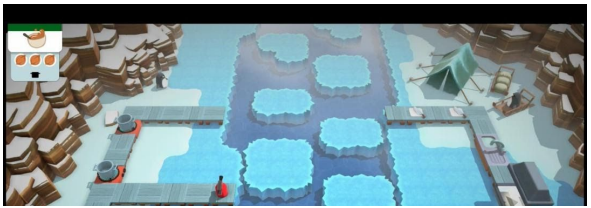


Inspiration

Crash Bash (2000)

Gang Beasts (2014)





Overcooked (2016)

Story

In hundreds of years, there are no humans on the planet. Have they gotten extinct? Or did they just leave the planet? No one knows. But in their absence, the mythical energy of the planet regained power and gave life to creatures such as… Ice Golems! However, these creatures soon realize that the world is not like how they know it. The weather is changing like crazy, no solid ground to stand, and what are those self driving metal monsters? This is no place for an Ice Golem! But their will is strong and they will do anything to survive. Even if it means to get rid of their own kin. After all, Ice Golems live in solitude.

Lobby

A player starts a lobby from the main menu. Additional players may join at any point in the lobby before the game starts (for the original version, only couch co-op is supported). A player may also start the game alone.

HUD

There is no real in-game HUD. Instead we want the players' focus to be entirely on the environment and the players they are up against, as well as the events taking place. There is HUD for swimming only in water, showing the time remaining before drowning.

Camera

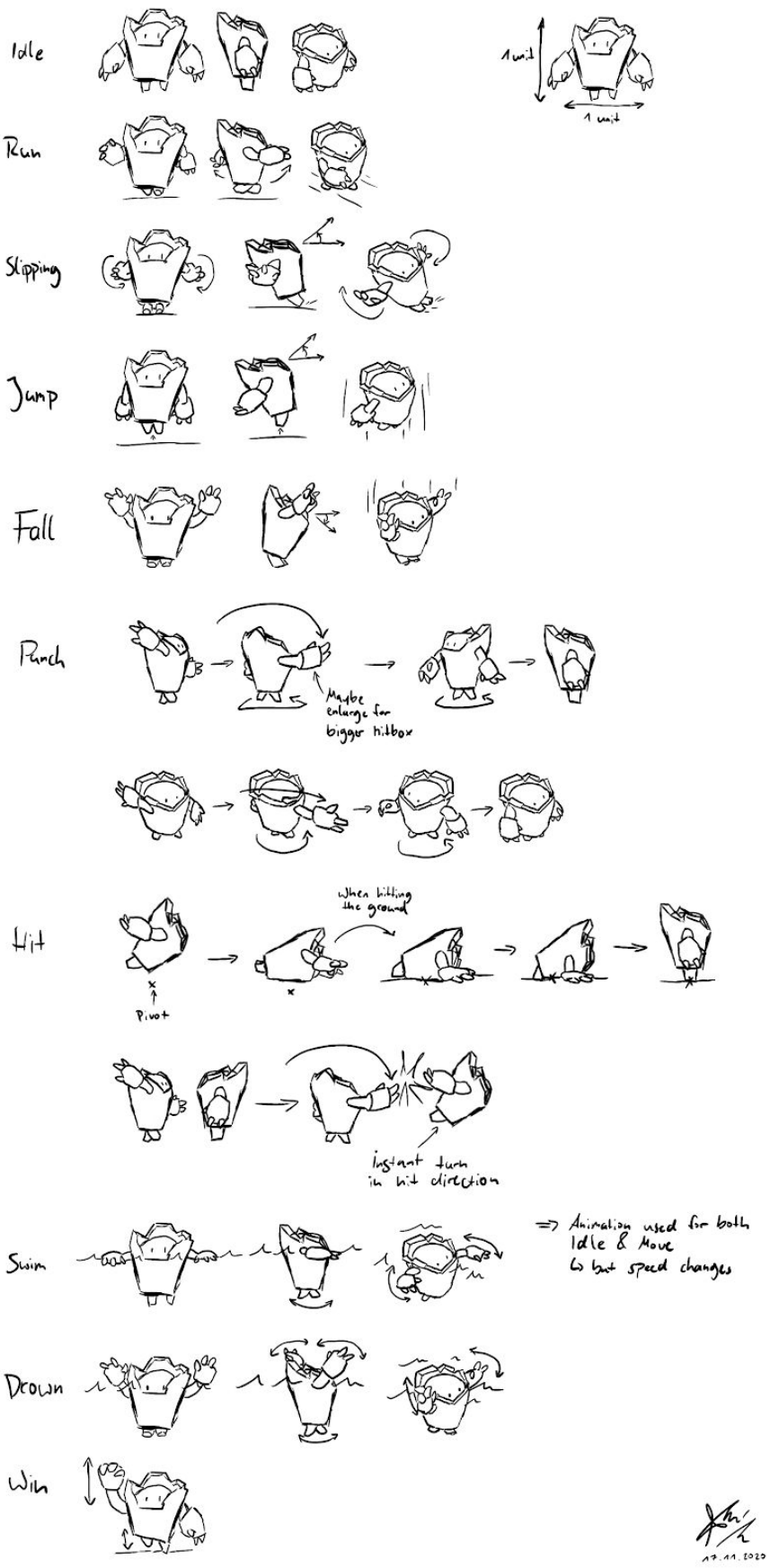
The camera views all players from a slightly tilted top-down view that puts emphasis on the characters and the hazards that are nearest to them. There is no split-screen, as all the players should be visible in the same view, which additionally allows for fairness, especially when considering game-changing events or environment destruction.

Graphics

The game has cartoony graphics with emphasis on special effects, and a cute artstyle that draws from the inspirations used in coming up with the game, as well as helps to deliver the intended fun message of the game, while putting players in the correct atmosphere.

Character Design

The Ice Golems that the players control take on a very close resemblance to the design of the entire game, resembling a very cute and wholesome interior that attracts the players, while having a pretty rough outer shell that deters enemies and keeps danger at a distance.

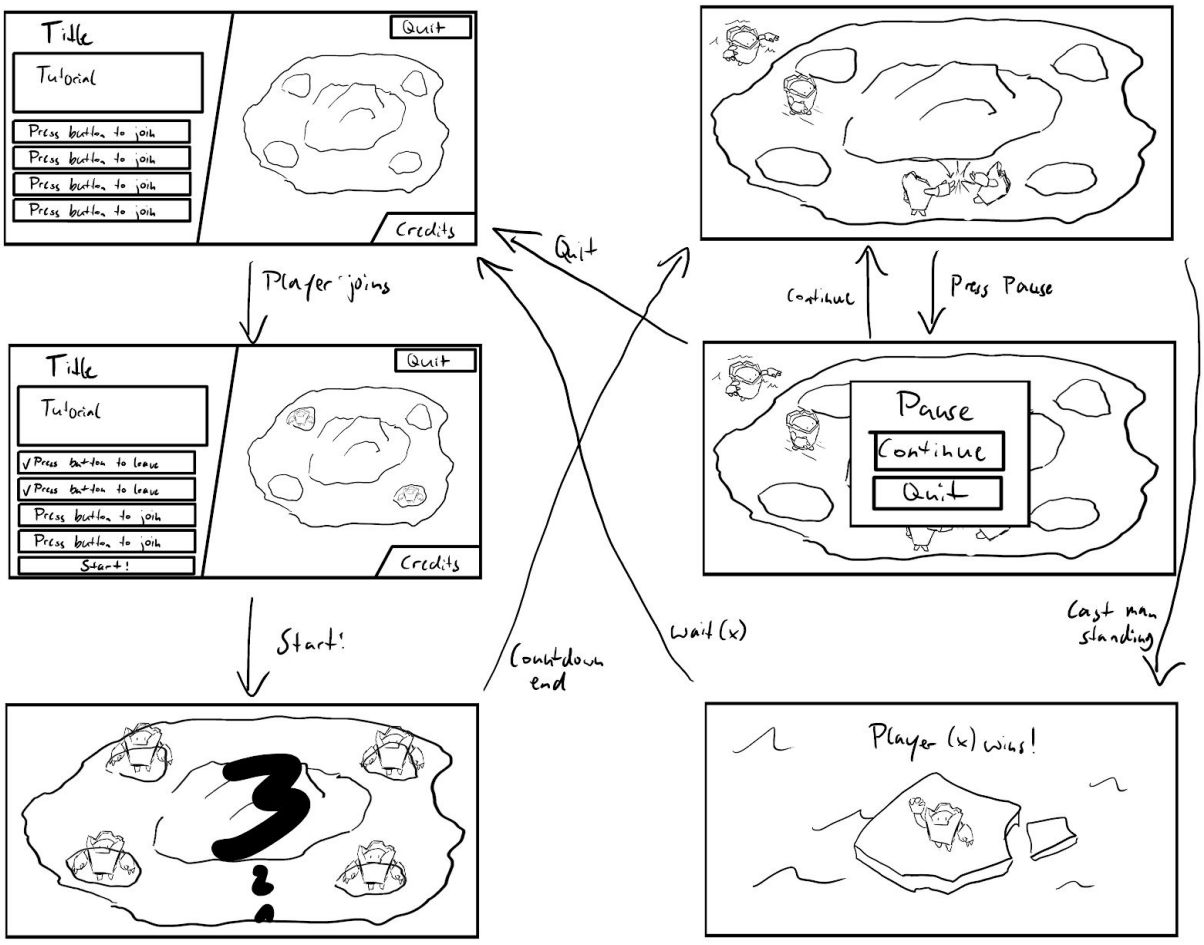

Idle
1 unit
1 unit
Run
Slipping
Jump
Fall
Punch
Maybe enlarge for bigger hitbox
Hit
When hitting the ground
Pivot
Instant turn in hit direction
Swim
=> Animation used for both Idle & Move
Drown
Win
17.11.2020

Sound Design

For the Background Music, different music for each ongoing event; they are very similar but also slightly different so that the players do not get shocked from the change, but rather enjoy a smooth transition that signals the different parts of the level.

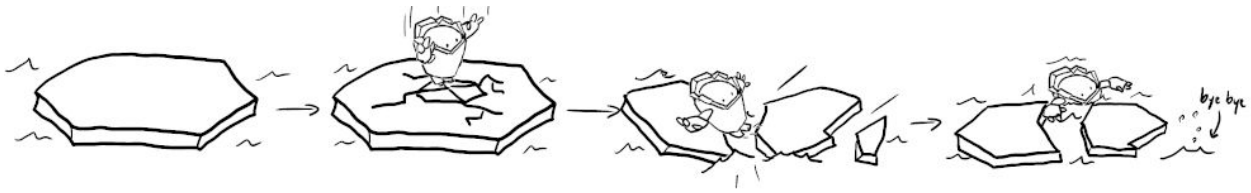
Basic Game Loop

Upon entering the game, the players are greeted with a countdown, after which the game starts and the players have control of their characters. As time progresses, the environment starts to fall apart, and events start occurring to make survival even harder. The platform takes on different physical properties depending on several factors (events, players, environment breakup, etc.). The last player standing is the winner.



Environment

The players start out in an environment that is very largely intact, but very slippery, making it hard for players to navigate the level. As time progresses and players move around, the environment breaks apart into multiple smaller platforms, making it hard to stay on the already slippery ground. The environment is also affected by dynamic events that determine to a large extent how the level is broken up.

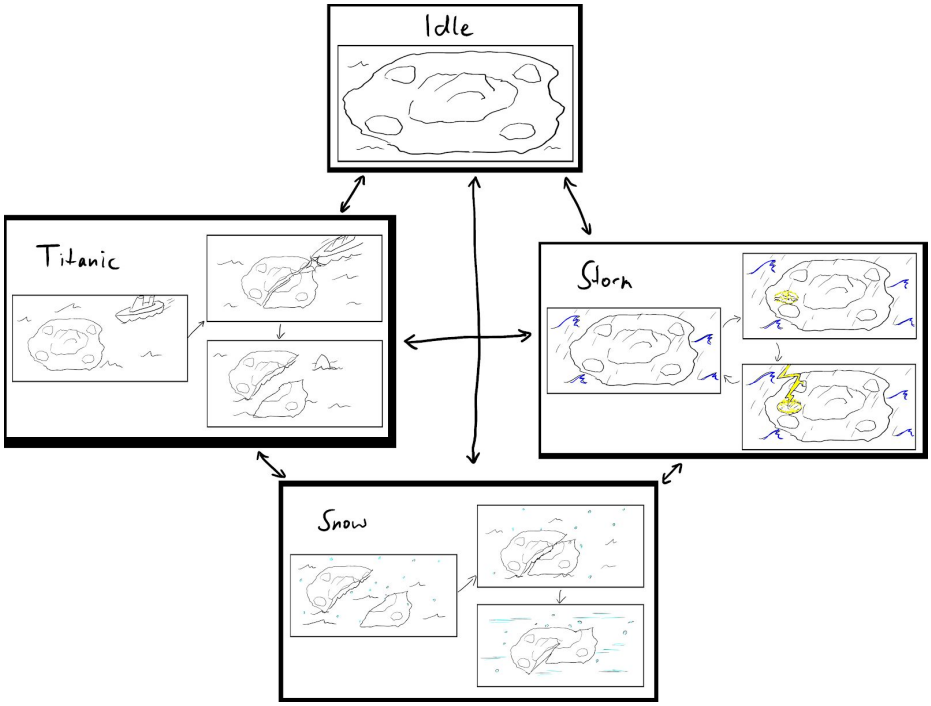


Events

The events are dynamically triggered at random throughout the gameplay loop. An event will occur, causing a change in the sky, water, sound(s), and can also carry some dangerous hazards that the players have to watch out for. The events alternate, occurring one after another in no particular order.

The events are:

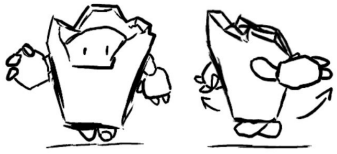
- Idle, where the tide is calm and the sky is clear
- Titanic, where a ship hits the stage causing destruction from the point of impact
- Storm, where lightning occasionally hits the stage, stunning players the affected area
- Snow, where the tide is flat, and water freezes to reform parts of the broken stage



Player Actions

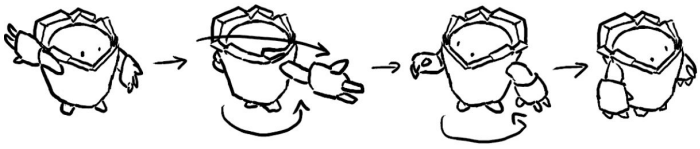
All players have the ability to:

- run, which is much harder on the snow



- jump, and they must land before jumping again

- punch, to try and knock players off the platform



- swim when they fall into water to try to get back to safety

PvP (Default)

In Player versus Player (PvP) mode, 2+ players are present when the game starts; they start out on a platform, each player on different end, and as they start playing and the environment starts to break apart, they must do their best to survive, while also trying to knock all other players off the platform, and making sure they cannot outlast them. The round ends when only 1 player is left standing with all other players drowned.

Solo Mode

In survival/solo mode, the player tries to survive the dissipating level and hazardous events for as long as possible, allowing players to enjoy the game even if they have no friends around to play with. The round ends when the player dies, and the timer's value is saved as their survival time (if it's high enough for a high score)

Leaderboards

Leaderboards allow players to compare scores and compete to climb the ladders, ranking them among their peers top to bottom. The PvP leaderboard ranks players with respect to time. The Solo leaderboard ranks players using survival time, with the player with the largest survival time having the highest ranking.

Assessment

The selling point of the game is that it is a physics-based competitive multiplayer game, which is easy to play and fun for gatherings and game nights. The game will appeal to people who enjoy couch multiplayer and party games. It provides simple controls and visuals that will make it enjoyable for a wide range of audience. The game also provides some fun mechanics like slippery physics, jumping, punching, buoyant platforms, and dynamic destruction system which when combined together will give a unique fun experience.

Development Schedule

Used development style : *Agile Development*

Sprints

- Physical Prototype (17.11-02.12) **1**
- Interim Demo (17.11-23.12) **2**
- Alpha Release (23.12-27.01) **3**
- Play Testing (27.01-09.02) **4**
- Final Release (09.2-23.02) **5**

Each Subtask is assigned according to its priority to the 5 sprints.

Akbar || Moritz || Tarek || Omar || ALL

EPICS	Tasks	Subtasks	Milestone
UI	Lobby UI	Design and integrate Title	**2**
		Design and integrate Tutorial	**3**
		Add credits	**4**
		Design and integrate Quit Button	**3**

		Design and integrate join/leave button	**2**
		Design and integrate Start Button	**2**
	InGame UI	Design and integrate Countdown	**2**
		Design and integrate Event indications	**2**
		Design and integrate Pause Screen	**3**
		Design and integrate Game Over UI (with and without winner)	**3**
	Character UI	Design and integrate Strength Bar	**2**
Character	Controller Input	Handle multiple inputs for multiple players	**2**
	Character Mechanics	Implement Moving	**2**
		Implement Jumping	**2**
		Implement Punching	**3**
		Implement Swimming	**3**
		Test and polish mechanics feel	**4**
	Character Physics	Setup gravity	**2**
		Make movement dependent on the surface (slippery or not)	**2**
		Setup Buoyancy	**3**
		Implement hit behaviour	**2**
	Character Animation	Integrate the animations to the character movement and state	**2**
Stage	Level Design	Create whole stage design	**2**
	Breaking Mechanic	Break one object into two	**2**
		Make the breaking depend on hit point	**3**
	Combining Mechanic	Combine touching multiple objects into one again	**3**
	Piece Physics	Add buoyancy	**2**
		Piece lifetime and sinking	**3**

Art Creation	Character	Modelling, Texturing and Rigging of the character	**2**
		Create Idle Animation	**2**
		Create Walking Animation	**2**
		Create Jumping Animation	**2**
		Create Swimming Animation	**3**
		Create Hit Animation	**3**
		Create Drown Animation	**3**
		Create Win Animation	**3**
	Stage	Modelling and Texturing level meshes	**2**
	Titanic	Modelling and Texturing the ship	**3**
VFX	Storm	Lightning spark effect	**3**
		Lighting strike effect	**3**
	Titanic	Smoke particle effect	**3**
	Water	Water with adjustable waves	**2**
		Foam on object interaction	**2**
		Frozen Water	**2**
	Weather	Rain particles	**3**
		Snow particles	**3**
		Lighting settings	**2**
Game Loop	Events	Titanic Behaviour	**2**
		Storm Behaviour	**2**
		Snow Behaviour	**2**
		Transition between events	**2**
		Update UI	**3**
	Lobby	Handle player join and create character	**2**
		Handle Quit Button	**2**

		Handle Start Button	**2**
		Update UI	**2**
	InGame	Implement countdown	**2**
		Implement Pause Behaviour	**2**
		Check for survivors	**2**
		Implement Game Over Behaviour	**2**
Prototype	Brainstorming	Find ideas how to implement a real world prototype of our game	**1**
	Creating	Create the prototype	**1**

Prototype

The game Slippery Bash is a competitive multiplayer game. Thus, it is very important to balance all its features, or otherwise, the game will quickly evolve to a one-sided massacre. There are a couple things that can be balanced within the game:

- Action availability
- Stage size
- Stage durability
- Difficulty of movement
- Defeating and recovery

Our goal of the prototype is to test these situations and see their strength and weaknesses.

Implementation of rules

The prototype will be physical as required. However, since it's impossible to play a dynamic and simultaneous situation on a board game, we need to convert most of its mechanics.

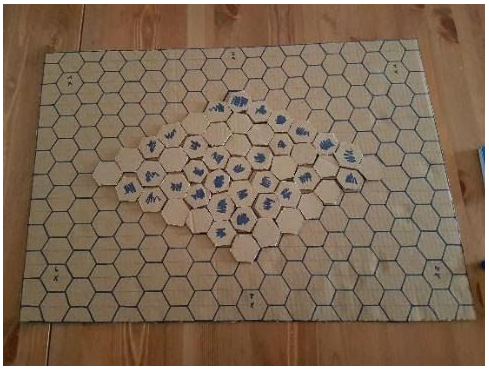
Dynamic stage but discrete movements

Difficulties of implementation

As in the game, the stage can be broken by jumping or collision with a ship. However, we cannot easily simulate waves and slopes on our board game, which is why we left this feature out. Also, the destruction of the stage is meant to be physically generic and affects the whole connected island. Since that is also difficult to simulate, destruction is only applied locally on a single position. So, the base of our board game is a hexagon patterned surface on which we create the stage with single hexagon tiles. The start positions of the stage tiles are marked on the board and forms a diamond made of 49 hexagon tiles.

Surface and Movements

The tiles also represent the position and distance on the stage. Only one player can be on one tile and actions can bring a player from one tile to another. What is more, the type of surface of the tile also effects on the player's movements which are described later.



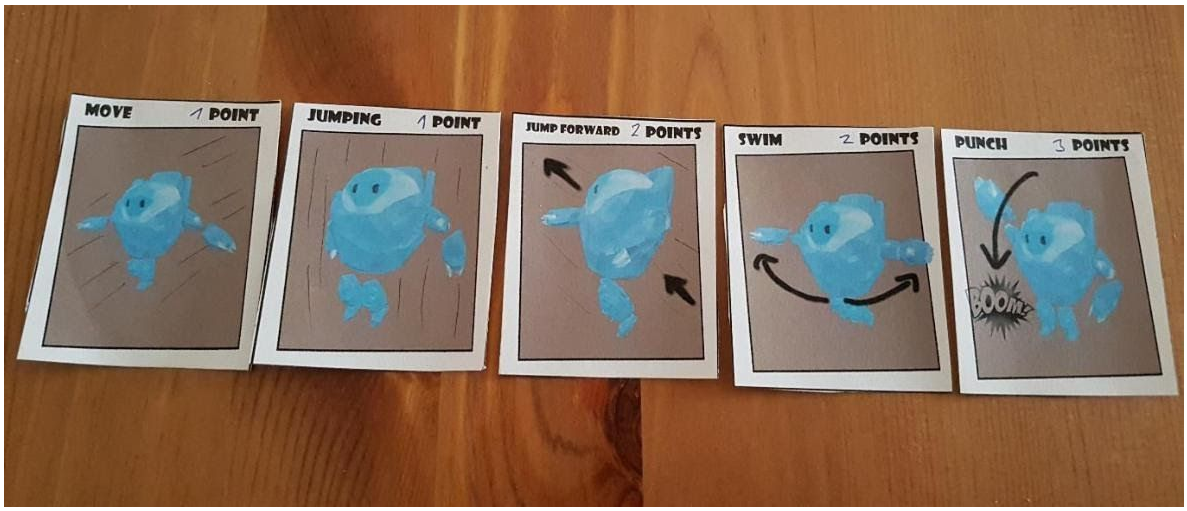
Durability

After all these changes, the feature of stage durability can be still tested. Each tile can be damaged and is then marked for cracks. For this prototype, we allow a tile to have 2 cracks before breaking on the third hit. A tile that is surrounded by water (no neighboring stage tiles) sinks on the next round after the last connected tile is gone.

Round-based Battle Royale

A round-based game is the closest we can do to simulate Slippery Bash. Each player can activate action cards only within their turn. The cards should represent the actions the players can do in Slippery Bash. In the actual game, these actions have duration that needs to finish before the player can do something else. So, within a period, a player can only do a certain amount of actions. That the reason why the action cards also have a cost and are only available if the player can afford it. For this prototype, we give the players 4 action points to spend them on these actions:

- Walk (1 point): Move to a neighboring tile. If the tile the character lands on has an ice surface, the character automatically walks another tile forward. The player can prevent this by spending another action point.
-
- Jump (1 point): Jumps on the same place. Mainly used for damaging the tile the character stands on.
-
- Jump forward (2 points): Move 2 tiles forward, no matter what is in between. Surface of the landing zone affects this action like on Walk. However, the landing zone gats a damage.
-
- Punch (3 points): Push a neighboring player 2 tiles back. The tile the enemy lands on gets a damage and can break right underneath the enemy. If the landing zone remains and has an icy surface, the enemy slides 1 tile further.
-
- Swim (2 points): Ice Golems are too lazy too swim. But they must, if they got pushed out from the stage and need to go back. This action card must be used when the character is on water in order to move 1 tile.



Events

For the prototype, we chose to only have the Titanic event because Storm and Snow require complex or random generations. This should not be a big problem of balance testing, since Storm is a threatening event and Snow a recovery one. In summary we don't miss a thing.

Titanic

Throughout the whole game, the ship is repeatedly crashing on the stage. At first, it spawns at one of the 6 start tiles that are marked on the board. To determine where, one player needs to dice the position. Then after every round, it moves a tile towards the stage. When the ship lands on a stage tile, the whole line of connected tiles in front of the ship sinks immediately. The tiles next to the ship also gets one damage. For the rest of the round, the ship sinks before it respawns at the next round.

We need an extra rule for spawning, since we cannot let the ship always spawn on the same 6 positions. Otherwise, the ship will eventually always go through without crashing. That's why, if the line of the start position is free of stage tiles, the start position moves to the closest line with a player on it. If there are multiple lines with this condition set, the players need to dice. The line is chosen which sum of the amount the players on it have diced is lower.

Gameplay

1. Beginning

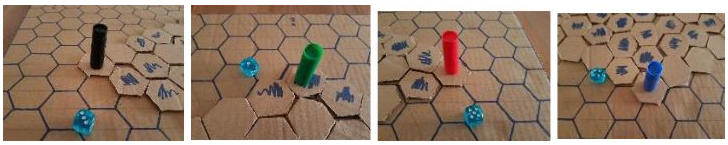
1.1. Lay all stage tiles upside down onto the marked tiles on the board.

1.2. Flip the stage tiles to reveal their surface properties.

1.3. Place the characters

1.4. Determine the first start position of the ship

1.5. Determine the order of turns



1.6. The highest number starts the main loop

2. Main Loop

    2.1. Player's turn: Use the action points to activate the action cards (not mandatory)

    2.2. Repeat with the player that had the second highest dice value

    2.3. **...**

    2.4. After the last player's turn: Iterate the Titanic behavior described above

3. End condition

    3.1. Only one player left

Testing and evaluation

Situation

Due to our current situation with COVID-19, we could not test our prototype in a proper manner. So, we tested it on our own and all the characters all played by one person. We hope of getting productive results, though.

Results

The game took about 2-3 hours with about 30 rounds. One reason for that might be that we constantly take notes and pictures of everything. Also, the tester is alone. However, it is still probable to play it at least for one hour in a normal situation. The idea of Slippery Bash is about playing short but multiple rounds together which can not be achieved with a game of that duration. Obviously, the reason for this is the balancing. So, let's go through all our focus points. In order to evaluate the result, we need a reference value that is compared with. We chose the distance a player can walk in one turn, which is regularly 4 tiles.

Action Availability

In a round-based game, every action makes a huge difference. Which is why most turns end up being a skip. It is always better to wait for the opponent to waste action points to come, so I can attack them with full power when they are close enough. In our prototype, the player can attack only opponents that are max 2 tiles away (regularly). Not very much for a quick attack. For the testing, we really needed to force "stupid" movements ourselves, so something happens. For improvements, maybe lower the cost of the punch actions to two, but restrict the player otherwise, like automatically skip after a punch. The other actions felt right in their availability.

Stage size

The stage size affects the balance between action and silent situations. If it is too big, there may be long silent situations before something happens. We aim for an active packed game, so our stage mustn't be too big. However, this was not the case because it took only 2 rounds before 2 players meet. So, the size of our stage is set correctly.

Stage durability

The destruction of the stage felt good. The importance of the choice of actions grew over time with the decreasing amount of time. There was, though, a time in the beginning where it's boring before the action starts. Maybe find a way to make the beginning more interesting. (In other Battle Royale games, for instance, the player is busy with looting in the beginning.)

Difficulty of movement

"Slippery and wet" is the theme we want to realize. We initially wanted to make things difficult with slippery surfaces. The icy tiles of our prototype, however, gives more advantage than disadvantage. While testing, we prefer walking on ice since they bring more distance. Only at the end when the stage is

small, it is quite inconvenient because you need to pay twice. Otherwise you might slip into water. But most of the time, these tiles are the player's favorite. Whether it's bad or not is unclear. Maybe we go to another direction with our design and it's good. Let's consider this.

Defeating and recovery

By this, it is meant how hard it is to kill an opponent, and on the other hand, how hard it is to go back to game after you've been punched into water. The result was: It is very hard! For both sides. This is the main reason why the game took so long. These Ice Golems just don't want to die. So, also for this situation, we had to force ourselves to do "stupid" movements to continue the game. But even then, the punched Golem is always able to swim back. Only at the end, when the attacker blocks the only tile that is close, one Golem finally drowns. Sometimes, we even need to destroy the stage on our own. On the other hand, it is still hard to recover. Even though you swam back, you're not able to evade the next attack. Then you get punched again and the loop continues. Until the perfect ship finally arrives to literally break the loop. So, we need a feature that will prevent the loop. A feature we've not yet found. But it's enough to improve the ability to kill. Thus, the total game duration is shorter, and the fallen players may rejoin.

Conclusion

Despite our doubts, the physical prototype has brought us productive feedback. Even though we had to adjust many things, the results apply also for the actual game. Also, many questions have been asked during our prototype brainstorming that needs to be answered for the actual game as well. So overall a good experience, which would have been better in a non COVID-19 situation

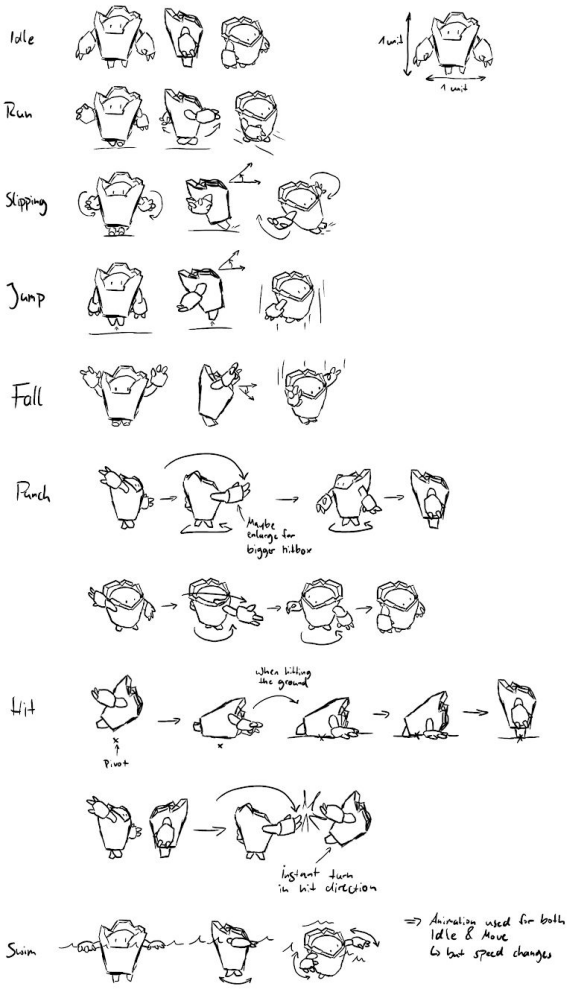
Interim Report

Overall Progress

Overall, we met most of our development goals for this milestone. The key missing factor would be polish. A lot of things are working, and working well. But the core of the game in a cleaner UI and final touches of cleanliness regarding some interactions. Some design choices also have not gotten enough playtesting yet.

Character

The character model was originally inspired by the Fall Guys characters with some more detail in textures and shape. The characters should look adorable and cute. The golem was modeled, uv unwrapped, textured and animated in blender. For animating the golem we used the rigify tool which is a helpful tool to easy animate humanoid characters. Having the issue that the Golem has no knee and only calfs and no thighs we wight painted the whole leg to the calves of the rig which worked quite nicely. The guidelines for the animations were similar to the modeling guidelines, it should look cute and adorable. We have the animations depicted to the right implemented. The implementation was done using the Unity animator, which is essentially a glorified state machine. The idle/run animation is determined by speed; the swimming is determined by a bool, and the other implemented animations use a trigger (i.e. punch, jump)



All the animations in action
https://youtu.be/JQ4A8x02aZ0

The model textured in Unity

Art

The art style is somewhere between cartoon style and looking realistic. We want to use the hdrp pipeline in unity to have very nice looking effects such as reflection and post processing features that really enhance the overall look of the game while keep some cartoon style to not have the need of modelling to high detailed models as well as keep some level of cuteness to the characters and world. The boat is a perfect example for this art style because it is very low poly modeled with only a view details but has some more advanced textures on it with cool looking reflections. For the future we plan to have also the tiles for the main island designed with this kind of art style and add more details as well as verticality to the level.

Controls / (Local) Multiplayer

The controls were implemented using Unity's new Input System. This allowed for the development to be much more focused on the integration of the physical controls into the actual movement and handling. This extends to the multiplayer, as the new Input System is capable of seamlessly handling different user inputs and registering them to different playable characters.

Camera

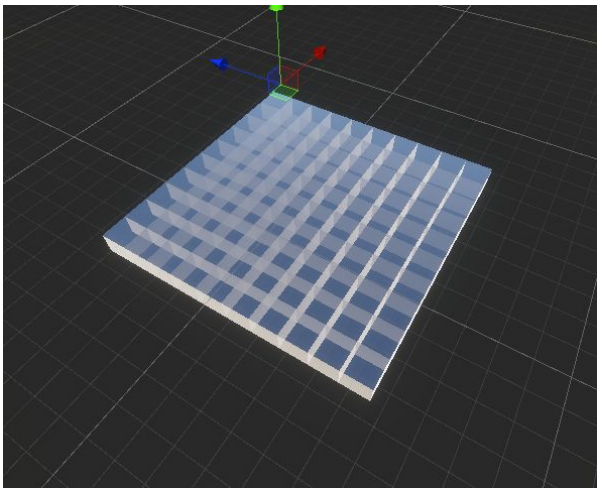
As per our design, the camera does not move, but rather shows. That means that placement was key, but with some minimal testing, it's clear that this would already be much improved if the camera had minor tracking that adjust the size and scale slightly to reframe the scene and keep all characters visible.

UI

The UI was more trouble than anticipated. We are well aware of the troubles that underestimating UI can bring, so it was already allotted its fair share of development time, but still it was quite unyielding, and often resulted in buggy behaviour. One of the biggest culprits was the “Start Game” button, which is technically a UI element, but due to how different it is (players can move around; the button for it is different than the default), we decided to handle it exceptionally to the other inputs, using a “virtual” button to trigger its OnClick method, rather than the UI’s built-in submit functionality. Once the very rough bugs were worked out, the UI got to a fairly stable state and definitely should not be an issue now.

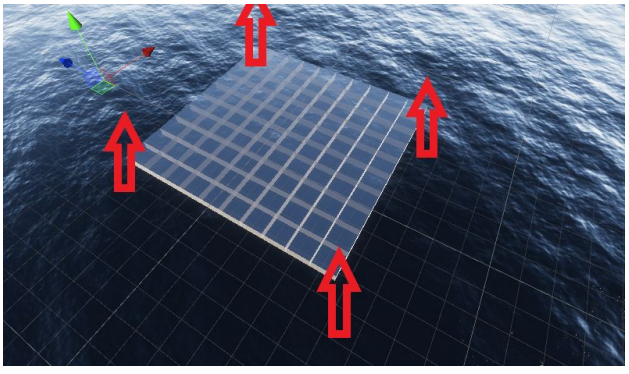
Physics

Platform



The physics system for the game was mainly built on the Unity physics engine, using simple colliders and rigidbodies (on the players) to support collisions, gravity, and physics-based movement. Other parts simulate buoyancy in slightly different ways (see buoyancy section below). In order to test the physics and mechanics of the game, we needed to create a simple platform. A flat cuboid was used to act as a platform. This cuboid is built up of 10 by 10 smaller cuboids (we will refer to them as tiles). So the tiles build up a flat surface that will be used as the main platform of the game where characters will be spawned on and where the brawl will happen. Each tile on its own is breakable and has a certain physical material. This allows us to easily integrate the different ice and snow (physics) materials (having different visuals and friction values) into individual tiles.

Buoyancy

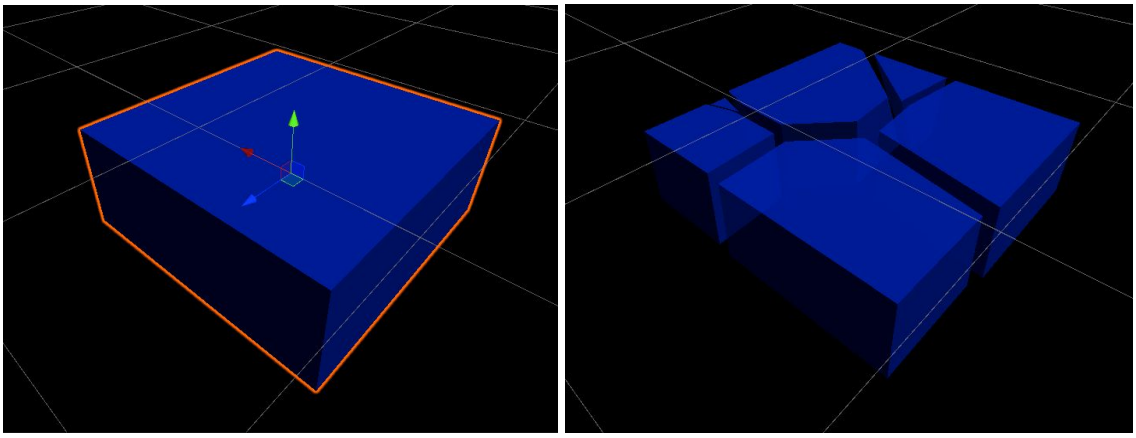


Since the platform will be placed on water, we needed to simulate buoyancy to it. At first, to achieve this, generic buoyancy scripts were used. But the result of these scripts were not good enough, as they caused the flat platform to be easily flippable and out-of-control. This behaviour, although can be realistic in some way, could be very frustrating for players as they can fall off easily from the platform and lose the game. So another mechanic was used in order to simulate the buoyancy of the platform. This mechanic aims to

make the platform not flippable but at the same time would have realistic floating movement that would give the desired difficulty of control. The idea of this mechanic is that instead of applying buoyancy to the platform as a whole with respect to its center of mass, we just add several forces to the corners of the platform to simulate the buoyancy. These forces act as thrusters that keep the platform a float with respect to the ocean level (which is dynamically changing), and these thrusters will only push the platform at their positions when they become below ocean level. The positions and directions of the thrusting forces are represented by the red arrows in the next image.

Destruction

As mentioned in the Platform subsection, the platform is built up of 10 by 10 tiles. These tiles are breakable, and they should be broken by certain events in the game. In order to implement the tile destruction mechanics, first a procedural mesh destroyer script was used. This script would randomly split the tile to a specific number of pieces that will be created as new meshes in the game. This mechanic was dynamic and gave random results which looked acceptable, but it proved to be very performance intensive which could result in frame drops when breaking several tiles at the same time. Instead of doing so, we used predefined broken mesh which was created using a 3D modelling tool.



As seen above, there are two states to the tile, the original one and a broken version which is split to multiple meshes. Upon activating the destruction of the tile, it is replaced with the broken version. After that a rigidbody component is added to each piece and an explosion force is applied on them to make them scatter away. In further iterations, we intend to have more intermediate meshes to indicate the damage level to players. Currently, it internally registers a set number of damage levels (without showing any indications of damage to the player) before suddenly exploding.

Character Physics

The character supports physics through a collider as well as a rigidbody. This allows access to all of the needed physics functionality. For the collider, a simple capsule collider was adjusted to the player's avatar; this is in contrast to a more complex collider as this allows us to save computations where they're unneeded (the physics we should be using resources to compute is actually outside the player model!). The rotations in the X and Z directions were frozen for the rigid body to allow movement through physical forces, without the 'capsule' toppling over.

Punching

An additional object that stores the player's punch location is stored, and, when the player punches, the C# code dynamically checks a radius in front of the 'punch', and applies a scaled impulse along the vector of the punch to the hit player.

Jumping

The jumping implementation is fairly straight forward. The player receives an impulse in their upward vector direction, and can not jump again until a collision is detected. At the moment, the ground does not have a proper tag set (look at known bugs section below), but it's unclear whether it would be better with just the ground refresh the jump, or any collision (needs playtesting maybe)

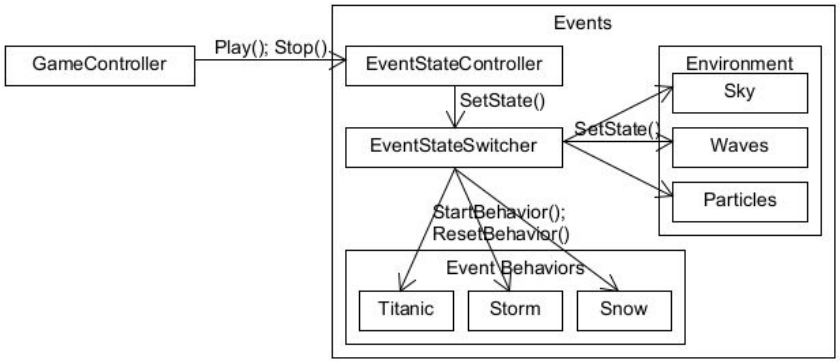
Swimming

Swimming for the player works very similarly to the platform; using the same buoyancy script (with different values), the player is kept afloat, wherein an upward force constantly acts upon the player to keep them at water level. One of the issues we faced was that the water has no real friction so where moving could simply apply a forward force to the player that is stopped by friction with the ground, this was no longer feasible for swimming, so we had to devise a new system wherein a velocity is 'forcibly' maintained as long as the user is 'moving' (through the controller). This could probably be improved.

Events and their Effects

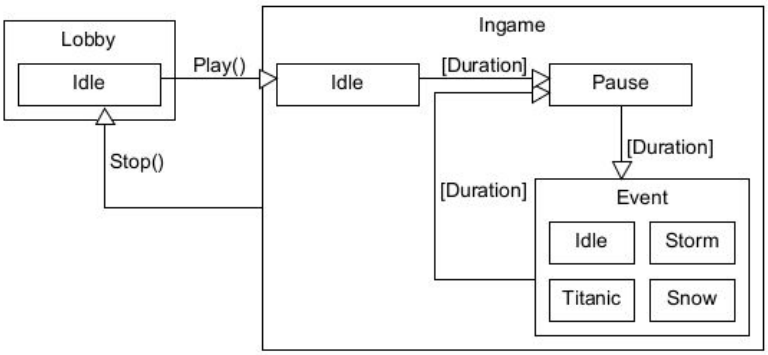
Base structure

The Events system has multiple subsystems and has one interface to the main Event Controller provided to the Game Controller. The interface provides starting and stopping the Events transition logic. The interface has access to the Events Switcher which handles the subsystems Sky, Wave, and Particles. Each manages their own states independently which are set by the Switcher dependent on the current Event. Also, the Switcher enables the current Event Behavior subsystem. Each Behavior subsystem is playable and stoppable by the Switcher.



Logic

The main logic of the Events is as follows:



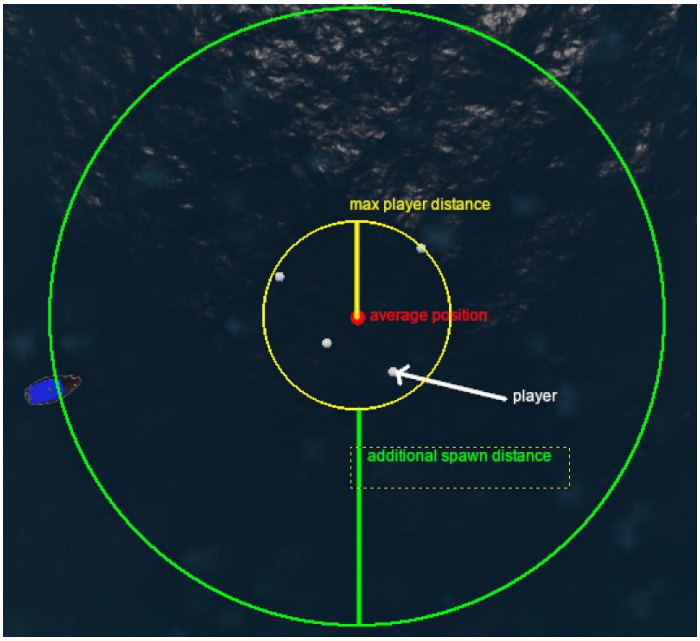
We most probably need to balance the event and transition duration and their probabilities, so they need to be easily adjustable. Also, we plan to change them over time which is why we also need to define the time after which the logic changes ends. For that, we use AnimationCurves as parameters so we can easily adjust the probabilities and durations over time.

Event Behaviors

The Behaviors are enabled by the Switcher if their respective Event is called and the transition to that Event is over. At the end of the current Event, the Switcher also reset the behavior again.

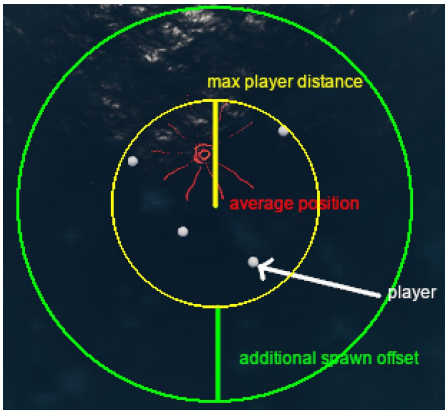
Titanic

The behavior of the ship depends on the player positions. The spawn point is set on a circle around the center of all players. The ship movement direction is set to the center and moves continuously in that direction. The speed is dependent on the current Event duration given by the Switcher. The destruction behavior is not yet merged with the Tile behavior. For now, the ship sinks when Switcher calls the reset.



Storm

Spawner: The behavior of the storm also depends on the player positions. It spawns Lightning objects out of the object pool within the circle around the center of the players. The amount of Lightning Strikes over the Event is currently constant, but a dynamic spawn amount will be added to be more flexible.



Lightning

The Lightning has three phases during its lifetime. Their duration can be adjusted:

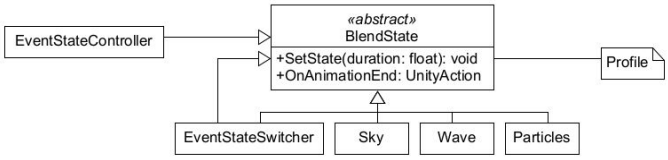
1. Ribbon: Indicates the Lightning Strike position on the water surface. Is made with the Ribbon feature of the Shuriken particle system.
2. Fall: Using Shuriken, multiple Lightning Trails fall from the sky onto the strike position. A script was needed in order to move all the particles to the point. The way they move until the strike can be adjusted.
3. Strike: A single Lightning Line Renderer pops up and the collider that hits the player is enabled for a short period of time. After that, the Lightning deactivates and is available again in the object pool.

Snow

Currently, only a collider is enabled during the Event. What is to be done, is also fixing the tiles and regenerating.

Blending Subsystems

All the subsystems Switcher, Sky, Wave, and Particles have states that transition smoothly between them. That is why all of their scripts inherit the abstract class BlendState.cs. This class defines the smooth transition between states. The inherited classes only need to define what is to be modified and the State Profiles for each state. The State Profile is a scriptable object that describes the properties, so we can adjust them easily on the asset.



Sky

Intensity of the sun, exposure, and indirect lighting are modified in this subsystem. There is no need of an actual sky box since our view is rotated downwards.

Wave

This subsystem handles the Ocean behavior. We use the asset *Ocean Crest* by *Wave Harmonics* that we are provided by the NGO *Cyan Planet*. This asset creates an ocean surface with height displacement whose waves property are modified by our Wave subsystem on runtime. The asset also provides a sampler to determine the wave height on a specific position, which we have adjusted for our purpose.

Particles

This subsystem changes the emission rate of two particle systems Rain and Snow.

Rain Decals

This feature is actually an Extra feature but has been implemented anyway out of technical curiosity. The challenge here was to spawn decals where a particle hits the ocean with good performance. As a result, we track all particles every [x] frames and check if their position is within a specified range around the current ocean height. If that's the case, a Decal Projector is spawned out of a limited object pool. The decal scales up over lifetime and despawns after that.

Integration

The integration was quite simple. As we proceeded with development, every feature (or set thereof) had its own scene, and for most features, when it was done, it could simply be saved as a prefab. We had a rolling main scene that we could then just load the prefabs into and attach the components as needed. For a lot of the singleton scripts (GameController, LobbyBehaviour, etc..), no references needed to be held as they used static functions that could simply be called as-is without reference.

Future Iterations

Possible Improvements

- Camera could move/scale slightly to make sure all players are kept in-frame as much as possible. Currently players can go
- UI as a whole still needs actual images and text; currently using Unity defaults
    - Lobby UI should be more explicit (“show” an actual lobby)
    - Pause Menu UI should be more explicit
- UI behaviour can still be more stable
- Determine clear jump refresh behaviour
    - Currently refreshes off any collision
- End-game behaviour needs a fair amount of work
    - Consider feedback regarding finality of death
        - Currently players do not can respawn instantly when they die
            - A feature not a bug?
    - No way of restarting game or going back to lobby

Known bugs at the time of the Interim Report

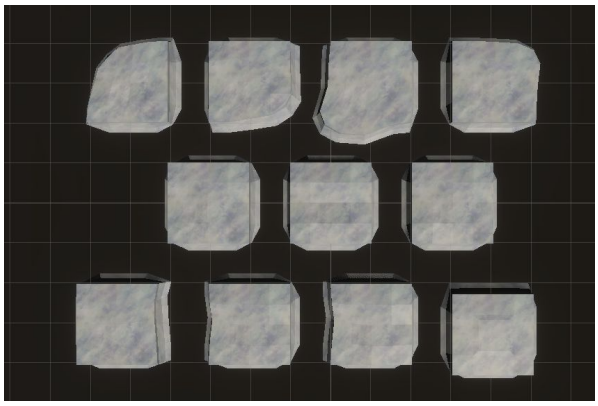
- Jumping does not properly damage ground tiles
    - This is due to the structure of the collider/triggers in the children/parents of the platform which does not correctly recognize the OnCollision Unity functions
- Swimming is fairly buggy
    - Currently no friction to provide real physics-based movement like on the ground
- Players can respawn right away after dying and no state is held to indicate this
    - This is a problem if there are more than 2 players in the game. Then the end-game condition of 0-1 players alive is never reached and the game continues indefinitely
- No damage indication on the floor tiles
    - This can make the game almost unplayable as seemingly perfect tiles will suddenly collapse under the players without any visual queues
- Light effects have some shading and illumination bugs

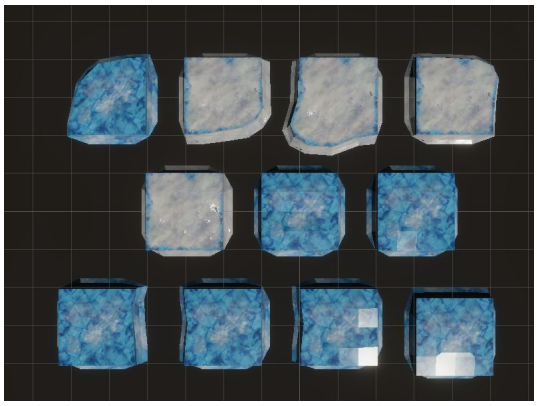
Alpha Release

Art

Level Design

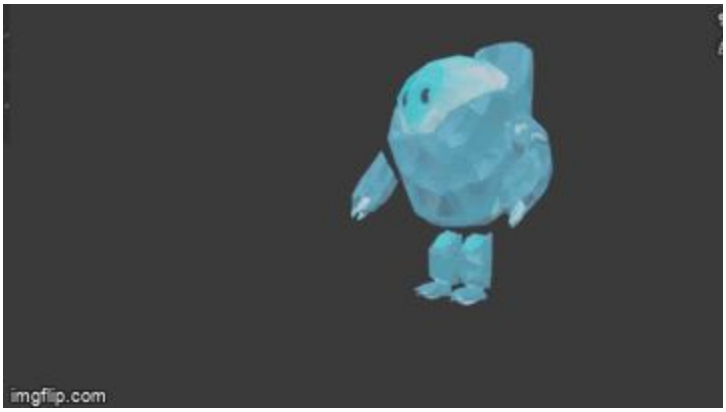
According to our paper prototype we planned to separate our island into small tiles which are then either snow or ice. That makes it easier to apply the slipperiness effect to the player character and due to the tiles system the island can be separated easily. However it was quite a challenge to have the tiles to have the snow/ice tiles blend nice into each other and also to make the tiles a little bit diversified. To accomplish this, we implemented a modular design for the tiles where we have 3 different types of tiles that are in the middle of the island with no contact to the ocean, 4 different types for each corner and edge tile. Each tile is individually fractured in blender. The random assigned material of the tile at the start of the game is then assigned to all broken tiles as well.





Character Animations

While testing the game loop we recognized that the hit animation was too slow and the impact of the animation was too weak. So we redone the animation completely and come up with the following:



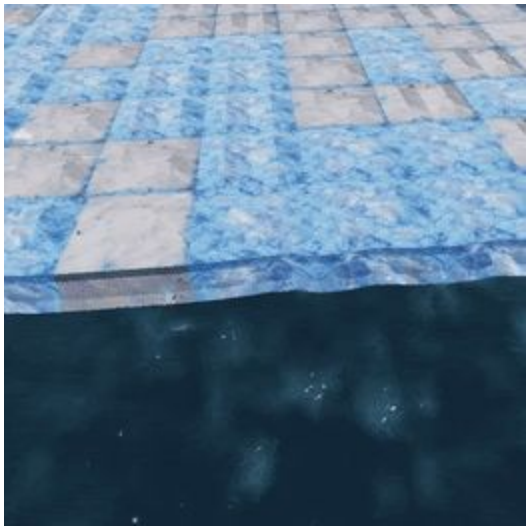
Platform Physics

More Tile States

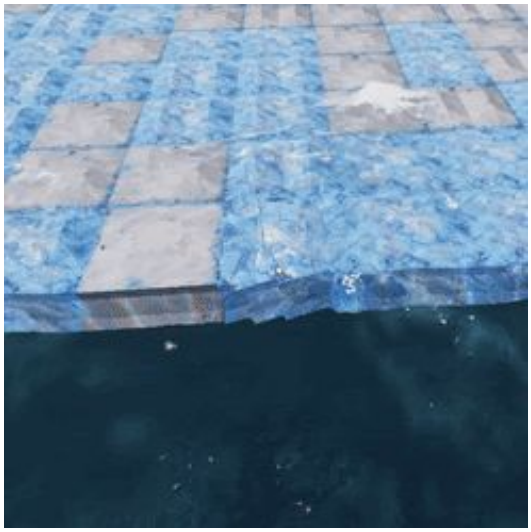
One more state was added to the tiles of the Island platform. There are three states now:

1. Normal tile
2. Cracked tile
3. Broken tile

All the tiles start as normal tiles and they take up to two hits. The first hit will crack the tiles.

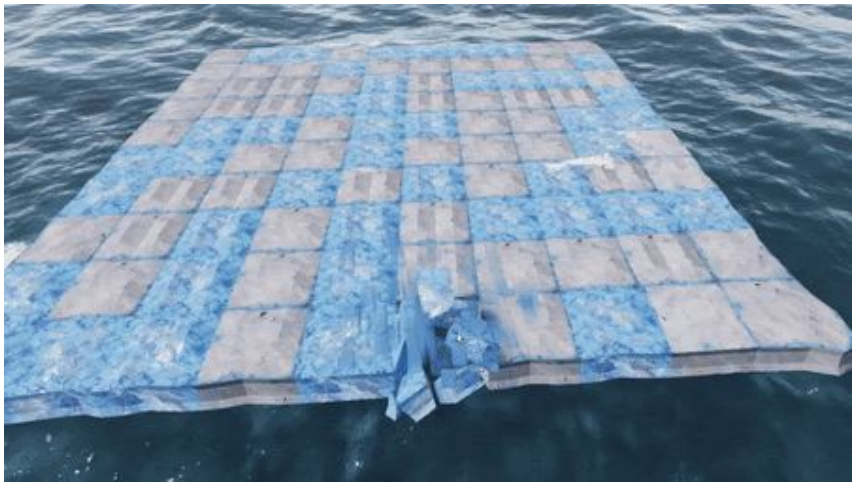


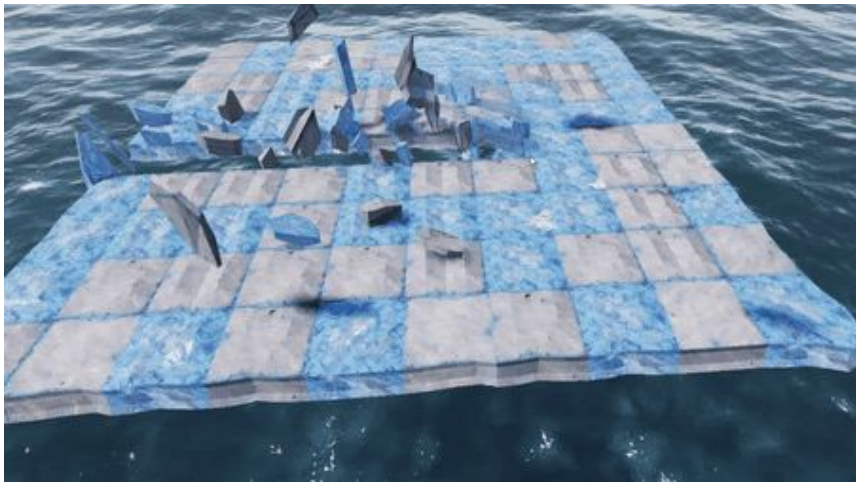
The second hit will break the tile into pieces and add an explosive force to them.



Dynamic Buoyancy for Sub-platforms

The game now supports dynamic destruction with separated buoyancy behavior for each sub-platform. Upon breaking any tile, the connection of the tiles is checked to see if there is any group of tiles that is separated from the main platform. In case there is, the separated tiles will be grouped under a new sub-platform and the sub-platform will be given its own buoyancy behavior. With this, a more realistic destruction system is achieved.



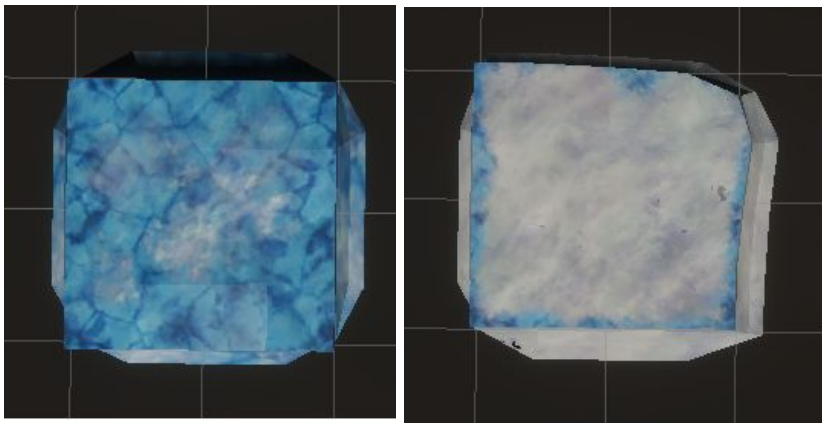


Physical Materials

Upon the instantiation of the platform, the tile type is randomly generated. There are two type of tiles:

1. Icy Tile
2. Snowy Tile

Each type of tile has its own visual and physical material which changes its behavior when the player interacts with it. The Icy tile is slippery and will cause the player to easily slide on it, while the snowy tile is more rough and gives the player more control when moving on it.



Sound

For the Alpha release, we added sound (music and sound effects) that should help with the user's immersion in the experience.

The sounds currently available are:

- Background music (that plays when no events are happening)
- Event-specific music that indicates the current event happening
- Player sound effects for the different action for clarity and immersion

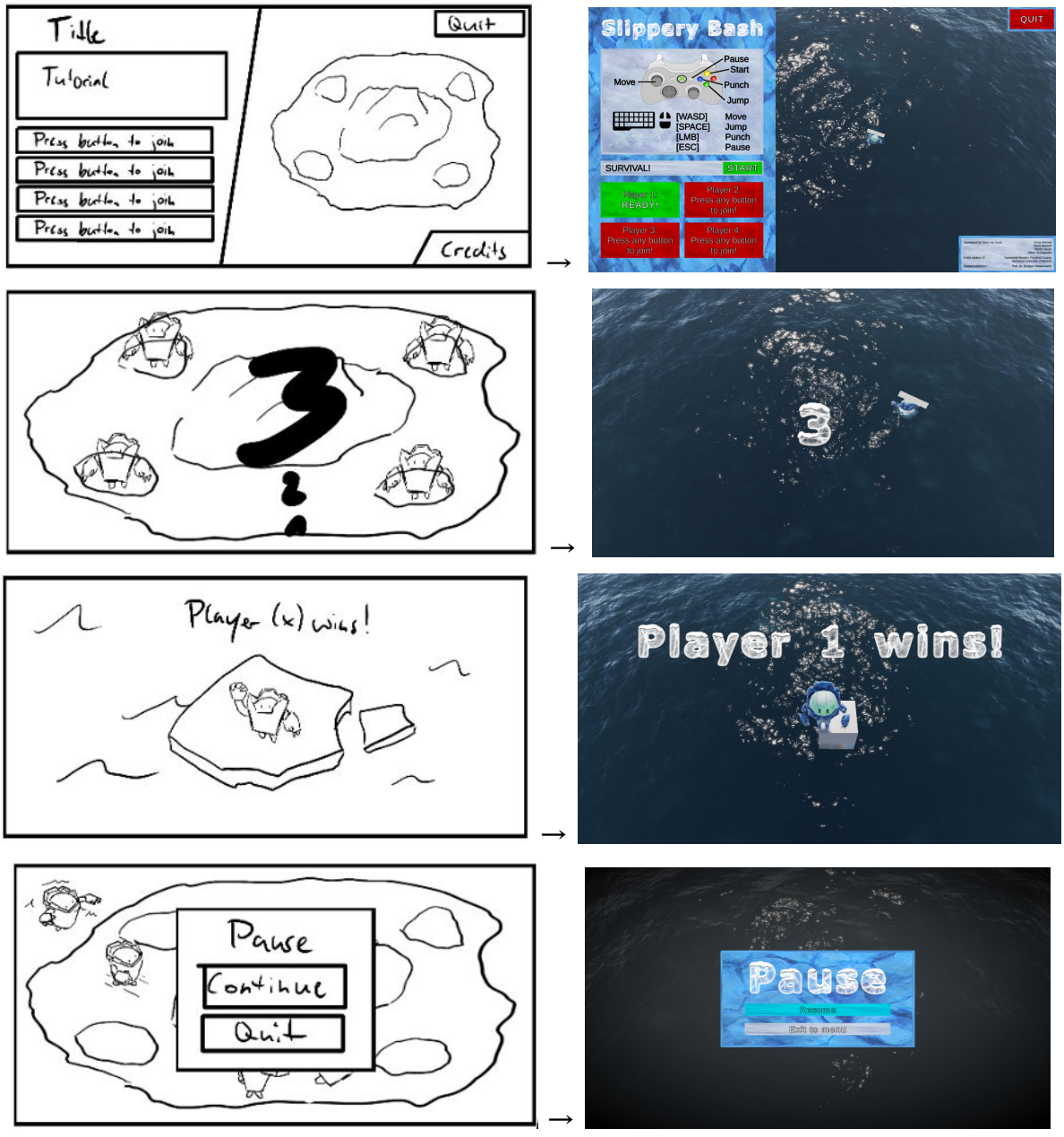
For the playtesting phase, we hope to gather feedback regarding

- The mixing of different tracks (i.e. attenuation levels)
- The quality of individual tracks (should certain tracks be replaced?)
- The placement of sounds (should something be removed? Is there something missing?)

UI

UI Visuals

The visuals have been updated that it fulfills the initial design. To fit the look of the game, textures that are used for the platform are also used for the UI panels. Furthermore, larger texts use a stylized image to look more “icy”. These text images are created via GIMP with a method described in this tutorial: https://www.saxoprint.co.uk/blog/ice-text-effect-in-gimp.



Additionally to the Pause UI, a vignetting and desaturating post-process effect is used to show that the time has been stopped.

UI Controls

At the end of this milestone, the UI controls seem stable and all users can access the UI independently (i.e. opening/closing the pause menu), but there is still a minor case of ‘not sure why it works’. Sometimes the UI control map seems to be enabled when only the Player (movement) map should be enabled, but it does work. For the playtesting phase we will try to clean up the UI further and look out for any inconsistencies the players notice.

Camera

In this milestone, we made the camera move dynamically during the game as the environment also changes. The position and distance of the camera now depends on the important object positions. These objects include:

- Center of the platform
- Players
- Titanic
- Lightning Strikes

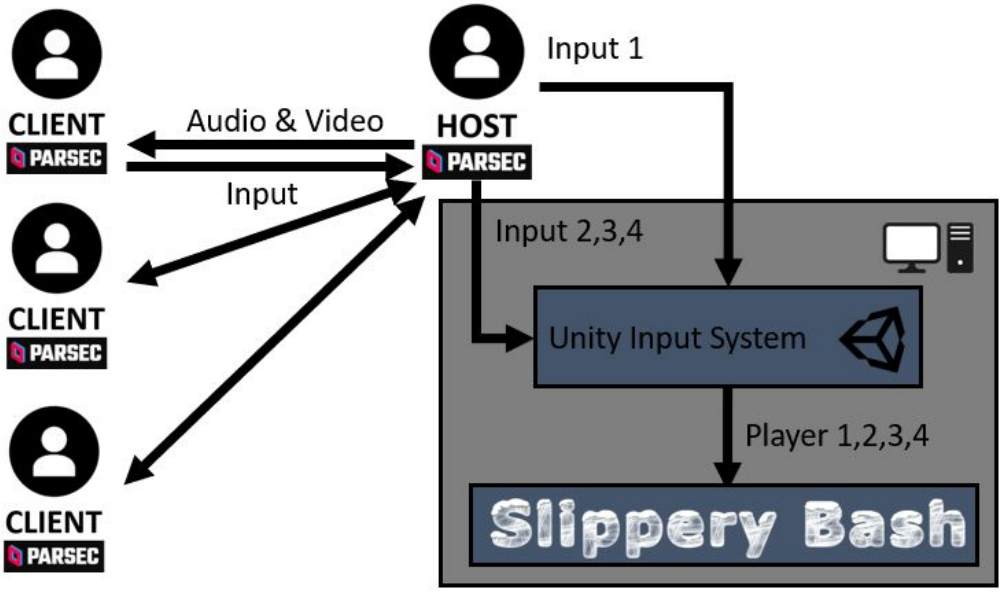
A bounds object is updated in each frame that encapsulates all of these objects. The camera is then moved smoothly so it focuses to the center of the bounds. The distance of the camera depends on the width or the length of the bounds, whichever is larger. This method is taken from this tutorial: https://www.youtube.com/watch?v=aLpixrPvlB8&ab_channel=Brackeys.

During the lobby, the camera is fixed so the environment can be seen on the right. And during the end of the game, the camera zooms in to the remaining player if there is any.

Parsec

Multiple players are mandatory for future playtesting. This is not possible during the current Corona situation which is why we needed an online multiplayer feature. Instead of adding an online feature to our

game, we used Parsec. It is a platform that easily enables us cloud gaming without additional code necessary. One player who has the game hosts their own PC, on which other players can join. The others get audio and video streams and share their input to the host. The host interprets these inputs as their own so the game itself behaves as if there were multiple local players on the PC. Combined with the local multiplayer already implemented in our game and thanks to the new Input System of Unity, we could instantly test our game together.



Physics

With the addition of Parsec, we were able to do a minimal amount of playtesting, and already figured out a few things that we are not happy with in regards to the physics. This will be one of the key factors addressed in the playtesting phase.

We will be looking to improve:

- Swimming (at the moment, this can get much too quick)
- Punch interaction in general (can feel slightly underwhelming)

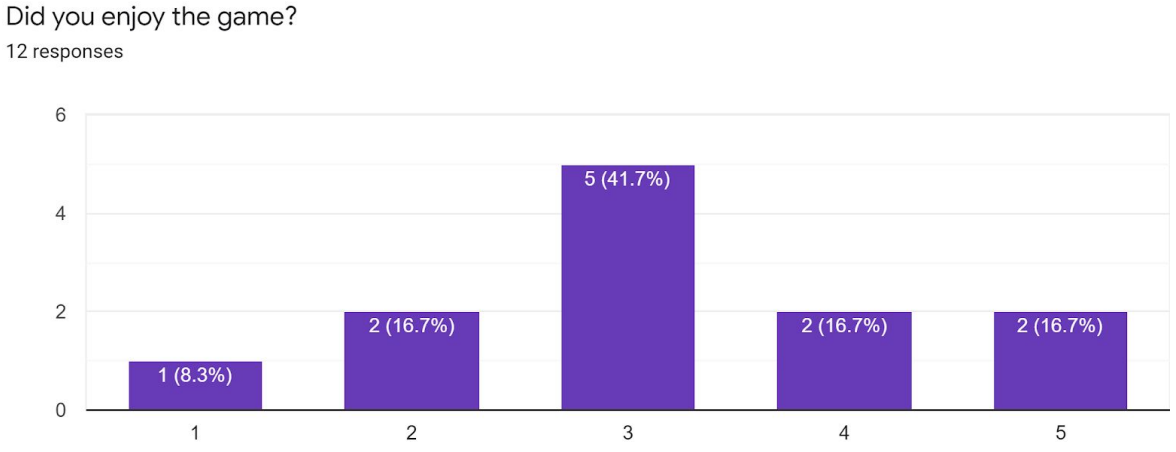
Other values and interactions might also be adjusted depending on the playtesting feedback.

Playtest Procedure

Set up of the playtest

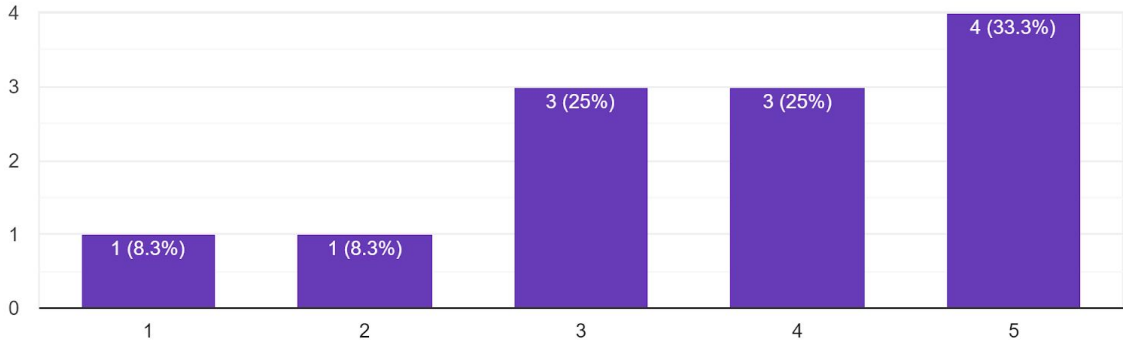
After we finished the Alpha release and played the game ourselves several times, we recognized that, before we hand our game to a small private group of testers, we have to fix severe issues of the game that can break the gameplay and make the game crash. Also our movement behaviour was too unpredictable so we redone it to make it more “controllable” for the players, we changed the input of the movement from completely force based to a more simpler approach. With the fixes finished, we then could move on to prepare the actual playtest. With our game being multiplayer, in the beginning we were worrying about being able to test the game. However, with the help of parsec it was very simple, and the testers just had to download parsec and we then were able to test the game with up to 4 players at the same time. To collect feedback, we prepared a survey with google forms, and every tester filled out the survey afterwards. During the playtest we recognized that we still have some bugs that prevent the players from finishing the game. After the first iteration we then fixed the bugs and also added some changes to the movement behaviour which then were tested in a second iteration of the playtest with general more positive feedback, however both iterations are summarized in the following survey to get a better overview.

Survey Results



Was the objective of the game clear to you all the time?

12 responses

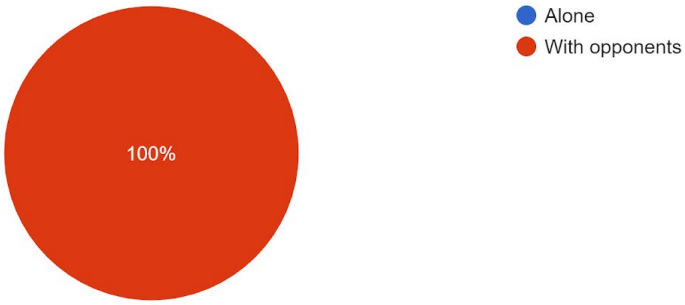


How was the length of one game round?

12 responses

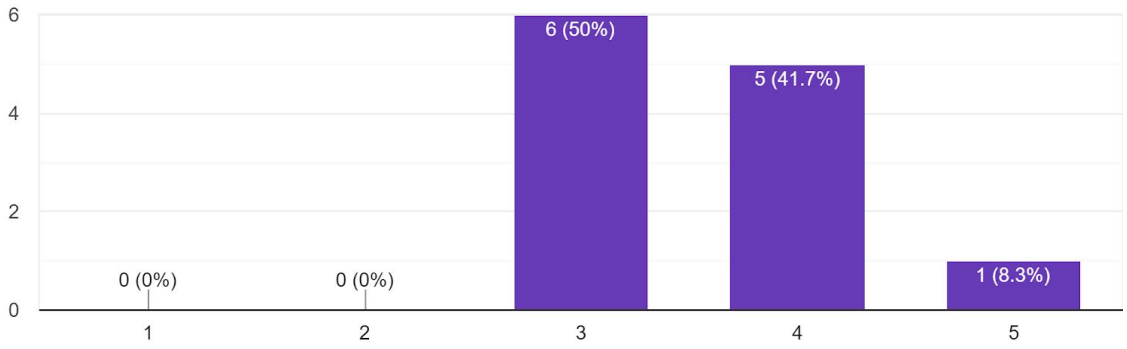
Do you prefer to play alone or with human opponents?

12 responses



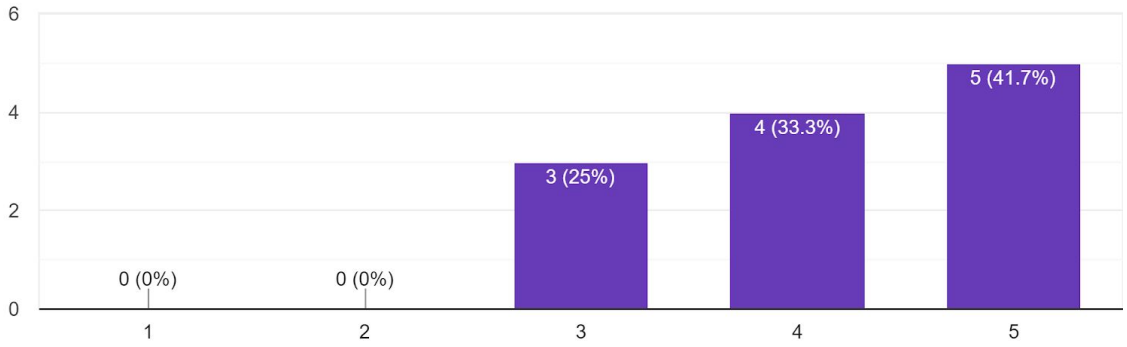
Was the game's premise exciting?

12 responses



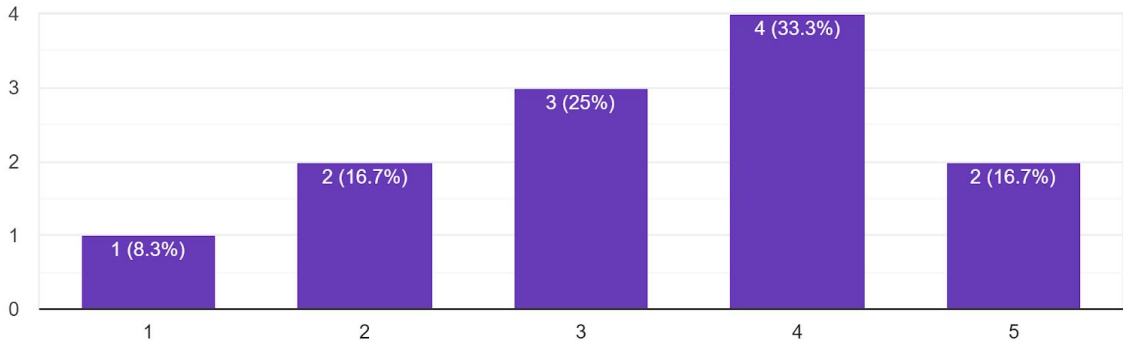
Is the game appropriate for the target audience?

12 responses



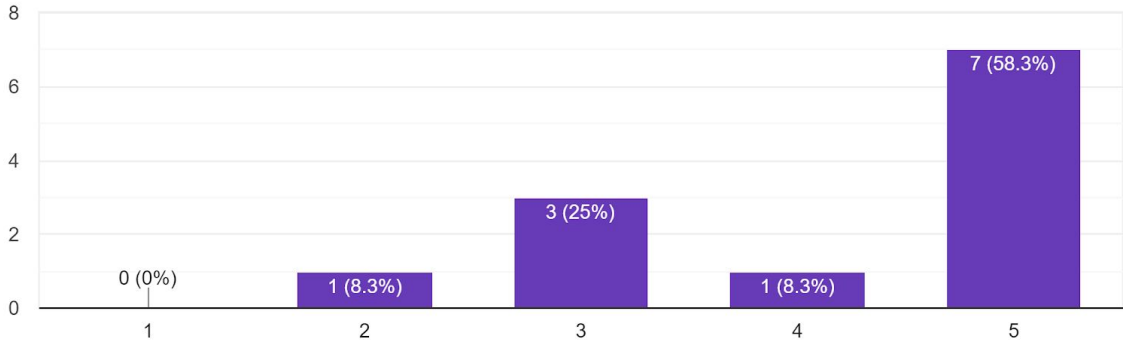
Did you feel a sense of dramatic climax as the game progressed?

12 responses



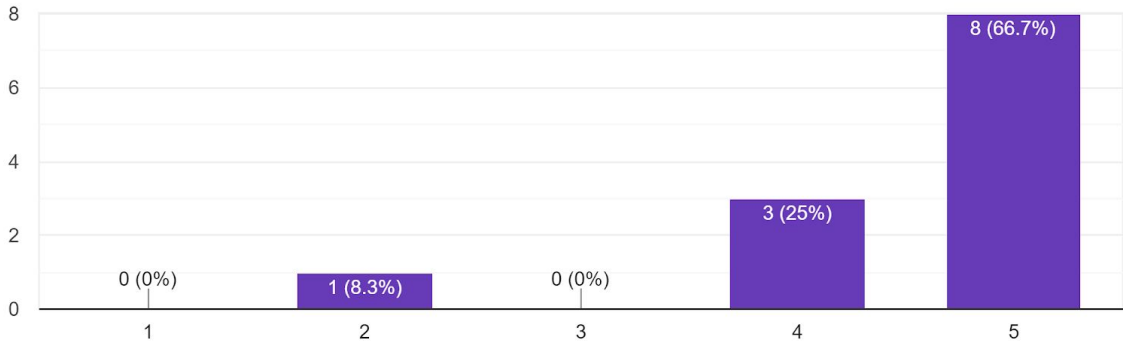
Where the rules easy to understand?

12 responses



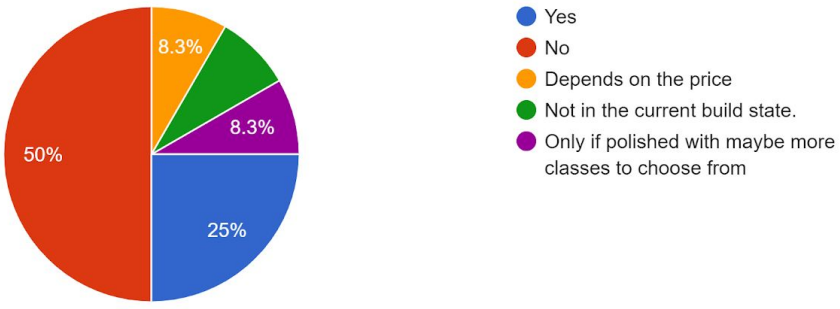
Where the controls intuitive?

12 responses



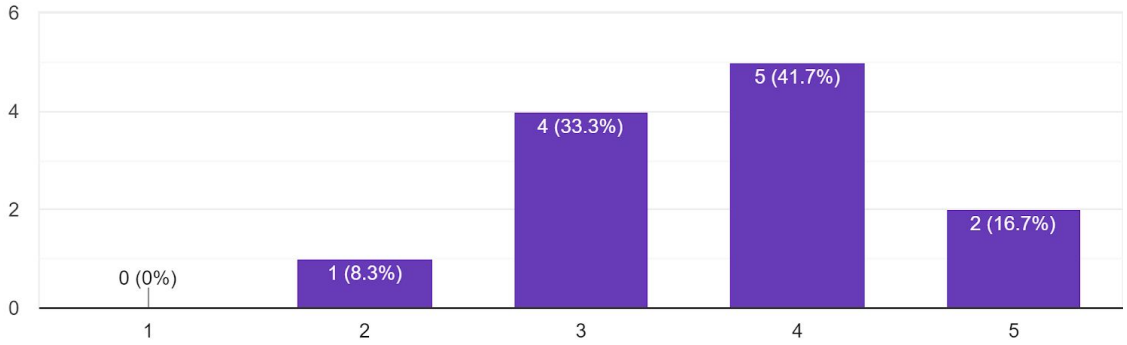
Would you purchase the game?

12 responses



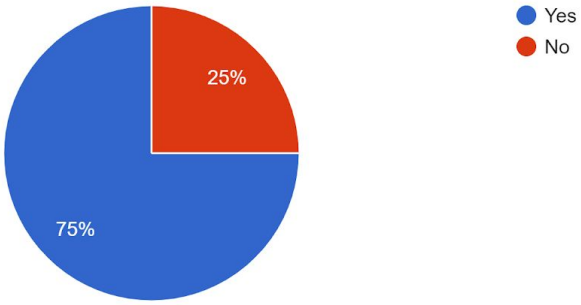
Did you like the art style of the game?

12 responses



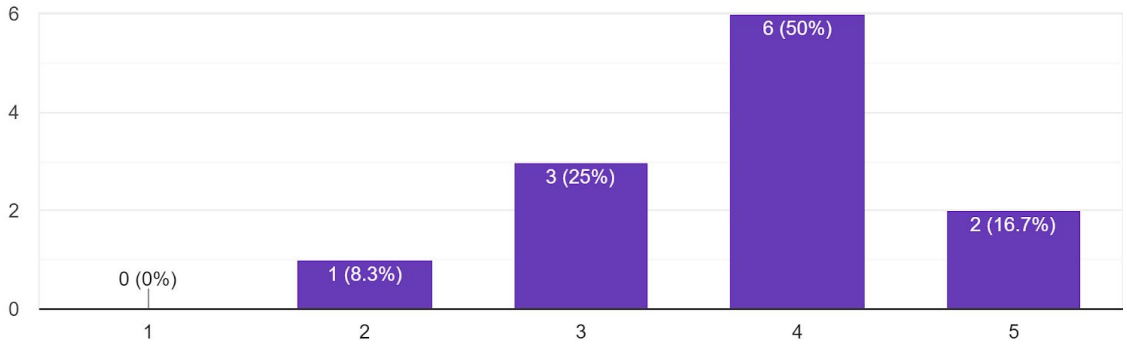
Would you like to play the game again?

12 responses



Do you think this game can be successful?

12 responses



Conclusion and Future Improvements

Most of the players were enjoying it, and also most of them think the game can be successful, which is first of all very important and great to know. Also the input system and the round length were from almost all testers liked. They also liked the art style and the sense of the dramatic of the game, however only about 50% want to purchase the game, so there is definitely something to add which we will discuss in the following:

Game Objective

As seen in the survey but also during interviews, we see that it was not always clear what is happening and what to do. Even Though the Menu UI shows the objective “Last man standing”, the testers didn’t know how. Also, it was not clear what the Events do except for the Titanic Event. That’s why we need to further explain them in the UI and other visual and audible cues in the final version.

Game Flow

Through the testing, it became clear that some players realized that just standing in one corner of the island is the best option to survive since there are no benefits of walking around and punching other players which increases the risk of falling into the water. Therefore we want to add a stamina bar that decreases with each hit of either player or lightning, and with reduced stamina bar the time the player survives in the water decreases as well. This way players are having better chances to be the last one standing if they try to reduce the stamina of other players by pushing them into the water.

As for the events, many testers found them too uneventful except for the Titanic event. To make the Storm event more interesting, we want the lightning to spawn directly at a player’s positions. Also we want to introduce a mechanic that makes it more likely that lightning strikes hit players that move less than other players, which should also motivate them to keep moving.

For the Snow Event, the testers expected the platform to regenerate, which was our initial idea that was discarded due to its complexity. To still keep a regenerating event, we introduce a new feature to our final version. An item will spawn during the Snow Event that will refill the stamina bar mentioned above when one player punches it. This should also increase the competitive motivation between the players.

Additionally, it became evident that jumping in the water was much too beneficial. We started by an implementation between testing iterations that disallowed the player from regenerating stamina unless they have actually landed on solid ground. This helped, but jumping was still too strong in water, so for the final release we’ve decided to remove the player’s ability to jump in water unless the jump can get them on solid ground.

Physics

As Physics is at the core of our gameplay, it is something that was iterated on multiple times before playtesting, and between iterations of playtesting, in order to meet player expectations and apply critical feedback. The highest priority was to allow the players to feel in control when moving their avatars, which was not previously the case. The version we used in the first testing iteration was using physics material to distinguish between the two physics behaviours and the avatars were moved via changing the velocity through input directly, thus the testers felt too chaotic, free of physics while moving and jumping. We have changed the movement behavior between the iterations, so the player is moved by force unless the avatar is standing on snow. The testers of the second iteration have found it challenging to walk on ice and easy to walk on snow, which we were aiming for. However, since the difference between ice and snow was very noticeable and the transitions were too frequent, it was hard for the testers to adapt to walking on ice and, therefore again, the players were not in control. That's why we want to have less snow tiles for the final version, so the players can better improve their slippery skills.

During playtesting, a physics-related issue regarding the platforms and level design was noticed. The issue was noticed when the testers found it hard to reach parts of the map as the broken platforms were moving far away from each other. This issue can cause an imbalance in the game as the size of the accessible platforms can be magnificently reduced if a big portion of the platform moved far away from the players. The issue also can cause the players who are on separate platforms to become very far away from each other thus affecting the camera view as well as the flow of the game. One way of fixing this issue is adding invisible walls which act as bounds to the map. Although this solution would keep the platforms from going far away, it will feel so artificial and unrealistic to the player. A better suggested solution is to make these invisible walls act as triggers instead of solid colliders, and when the platforms enter those trigger areas, they would be gentilly pushed back to the center of the map ensuring that the platforms are always close to each other and within the player's reach.

Audio

The audio for the game was clearly lacking as indicated in the qualitative feedback from multiple testers. The background music often felt repetitive and out-of-place. The music for one of the events (the snow event) is misplaced (rain sounds), and a lot of effects are missing, i.e. movement, swimming, lightning wind-up. This resulted in reduced immersion.

Some examples from the individual comments of the survey

"jumps and bashes are a bit extreme. dangers aren't very threatening. we can survive if we keep jumping. ice kept running away from us eventually we gave up the chase and just died."

"Maybe more events and the game be score/life/time based so to have a certain end to the round/game."

"Nice Game. A lot of fun battling with opponent. Especially when game proceeded and the ice flow was separated in several pieces."

“More explanation of events/ better visualization when platforms under water/ life bar always visible/ stronger attack if you hit somebody while jumping”

“Not many options right now. The premise is nice, need more polishing and game options.”

So overall, there were many comments on how to improve the game in future, mainly by polishing and fixing bugs and the by adding more features and variety to the gameplay as well as pointing out things that are happening in the game clearer to the player.

Final Release

Last Changes

The final version focused on final optimizations and resolving issues found through the previous playtesting milestone. These include mainly the new stamina bar, an invisible border that ensures smaller ice blocks don’t swim too far away, a modified snow event that has higher clarity and a crystal that utilizes the reworked stamina bar mechanic, an improved swimming vertical behaviour that removes independent player jumping and utilizes instead an automated jump system to help players get back to land when within distance.

Stamina Bar

The stamina bar was reworked as discussed in the playtesting chapter. The reworked system includes a full-by-default stamina bar (renamed from strength bar) that is dynamically affected by player interactions to provide a more interesting and unique playthrough every round, as well as playing into the new crystal system for the snow event.

Snow Event

During playtesting, the players found this event too uneventful. Since we have discarded the regeneration of the tiles, we added another regenerative feature. A Magic Crystal appears for a while during the Snow Event, and the Golem who punches it gets its max stamina bar refilled. Also, we brightened up the scene since there were complaints about that.

Platform Bounds

A problem noticed during playtesting was the platforms could swim far away from the center of the level causing several problems, including making some platforms unaccessible and players unreachable to each other. The problem is now fixed by adding invisible trigger bounds to the level that push the platforms back to the center of the level whenever they access the bounds area.

Tutorial

It was sometimes unclear for the testers what the mechanics and goal of the game were. Which is why a tutorial was necessary. We redesigned the main menu, so that the first player now can navigate through pages of tutorials. These tutorials explain the goal of the game, controls, tips for survival, and the behaviors of the events.

Lessons Learned

- Physics destructions are nice but to do it completely physical based is too compute intensive
- Pre defined destructions looks equally good and way more performant
- Water physics are cool but also difficult to tweak right
- To make get the right balance between a slippery surface and still controllable movement is quite challenging
- Local multiplayer is easy to implement
- Parsec is a great possibility to play “locally” but still physically separated