Soaper Duck

Game Design Document

1. Formal game proposal

1.1 Game description

Soaper Duck is a 3D endless runner where the player finds themself trying to progress as fast and as much as possible while navigating on a soap resembling vehicle. They have to successfully maneuver despite the low friction handicap that's put upon them.

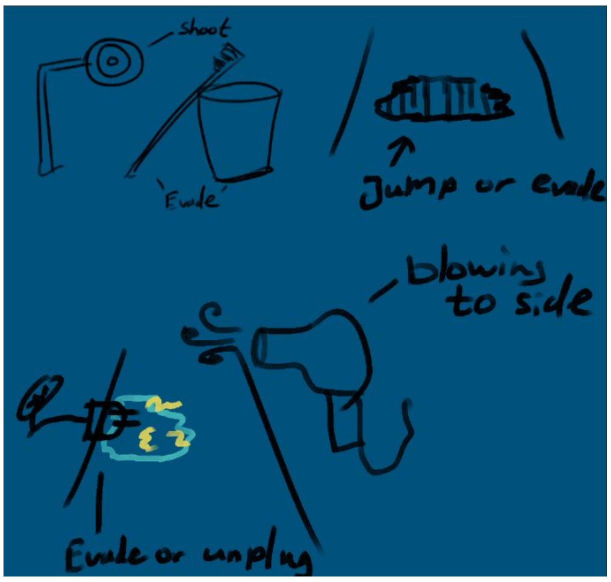
1.2 Core gameplay elements

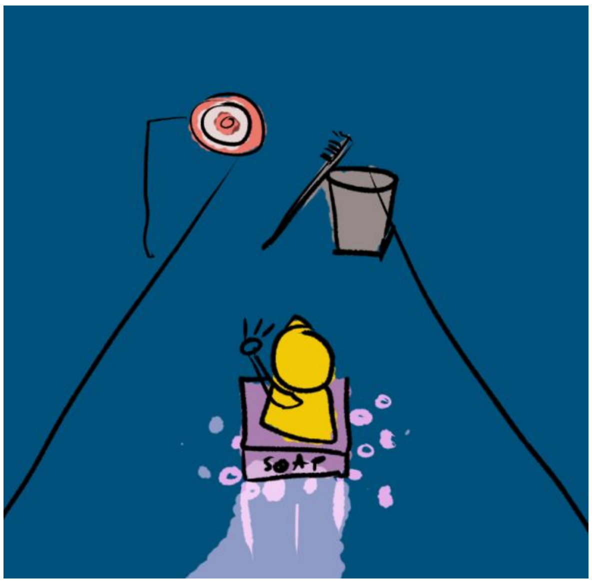
- 2.5D
- Main theme revolves around sliding on soap and water mechanics
- Endless level Generation
- Driving Controls
- Duck that can shoot water
- Triggers, Switches, Power ups
- Enemies

1.3 Features

1. Basic Features
   a. Score
   b. Soaps wears out
   c. Power Ups
   d. Terrain attributes, like wet ground having less friction
   e. Procedurally generated levels become more difficult over time
2. Advanced features
   a. Enemies
   b. Toggles & Obstacles
   c. Duck shoots water

1.4 Gameplay concept art





1.5 Character concept art

1.6 Big idea and technical achievement

1.6.1 Procedural level generation

Our procedural level generation is based on different types of building blocks that are combined in different ways to form a playable level. Levels become longer and more difficult over time by adding enemies, switches, triggers spawn at different locations within the level.There will be random special events like a shark attack that the player has to face.

1.6.2 Special effects

We might use metaballs for foam or other kinds of particle, shader effects to make the game environment more immersive and realistic. The duck can use it’s water shooting ability to make the floor in front wet, which will increase its speed.

1.7 Development schedule & Timeline

1.7.1 Layered development schedule

1. Functional minimum
   - Simple Player Character, that can move and jump
   - Start Platform
   - Playable Looping Test-Level
   - Game Over Screen
   - Input Manager (Controller/ Keyboard)
   - Simple models

2. Low Target:
   - Simple procedural Level Generation
   - Simple Menu UI
   - High Score (Points / Distance)
   - Restart Button
   - Visual Cues for Soap Mechanic
   - Scoreboard
   - Duck can shoot water

3. Desirable target:
   - At least 5 different spawnable obstacles in the procedural Level Generation
   - Background Music & Sound effects
   - Duck model for the Player
   - Different properties for Soap Vehicle and/or Floor
   - Power ups
   - Interactable Objects

4. High Target:
   - Playable without crashing
   - Multiplayer
   - Different Models for the Player to choose from
   - Different Sets of Level kinds to choose from
   - Character Customization
   - Seeds for level generation
     i. Ghost silhouette of previous runs for self competition
   - Save Replays
   - Watch Replays
   - Boss Fights

5. Extras:
   - Mobile Support

    - Twitch Chat Integration

1.7.2 Milestones and tasks

Milestone: Game Idea Pitch (18.11.2020)

Names Shortened as follows:

Albert Zach
Marco Grasso
Sahin Er
Sebastian Walchshäusl

Task Name	Who	Time (in Hours)
Brainstorming	Everyone	2
Documentation start	Everyone	3

Milestone: Physical Prototype (02.12.2020)

Task Name:	Who	Time (in Hours)
Defining of Prototype	Everyone	2
Creating a physical Prototype	Everyone	4
Documentation & Presentation	Everyone	5

Milestone: Interim Results (23.12.2020)

Task Name:	Who	Time (in Hours)
Simple Player Script → Movement & Jump	Marco	20
Playable Looping Test-Level → Start Platform & Simple Track	Sebastian	6
Simple UI → Start Screen & Game Over Screen	Albert	6

Assets & Models → Duck & Soap model, track parts	Sahin	6
Controller Support	Sahin	2
Simple Procedural Track Generation → Simple track building blocks (soap/bathroom styled)	Albert, Wacken, Marco	24
Infinite generation	Sahin	8
Documentation & Presentation	Everyone	5

Milestone: Alpha Release (27.01.2021)

Task Name:	Who	Time (in Hours)
Duck model for the Player	Albert	2
Soap model for the Vehicle	Albert	2
Interactable Objects → Switches, Triggers, ...	Sahin	4
Hair Dryer that blows player to the side	Sahin	1
Holes the player needs to jump over	Sahin	1
Power ups	Marco	2
Collectibles	Marco	1
Music & Sound	Sebastian	6
Advanced Procedural Track Generation → Include Obstacles & Collectibles	Marco	4
Different properties for the Soap Vehicle and/or the Floor	Sebastian	4
Documentation & Presentation	Everyone	2

Milestone: Playtesting results (27.01.2021)

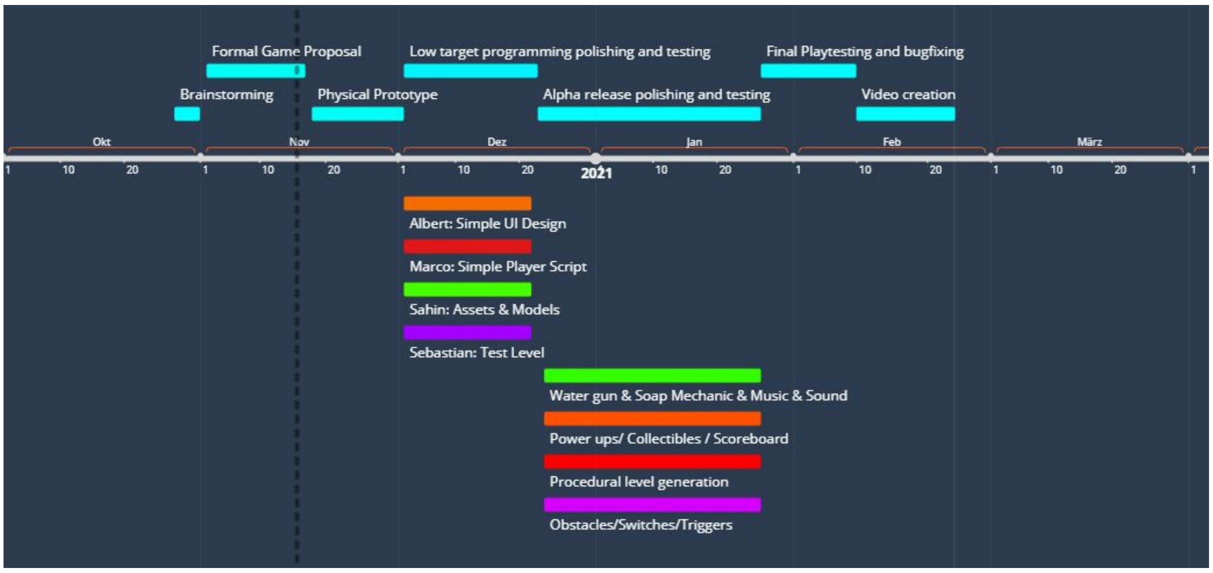
Task Name:	Who	Time (in Hours)
Testing and evaluation	Everyone	10
Bug fixing	Everyone	4

Implement Feedback	Everyone	8
Documentation & Presentation	Everyone	2

Milestone: Final release (24.02.2021)

Task Name:	Who	Time (in Hours)
Defining of Prototype	Everyone	2
Creating a physical Prototype	Everyone	4
Documentation & Presentation	Everyone	2

1.7.3 Timeline



1.8 Assessment

Tell us what the main strength of the game will be.
A challenge to beat your highscore and high skill ceiling.

What part is going to be the most cool?
Sliding as a rubber duck on a soap bar, while sniping targets with a beam of water.

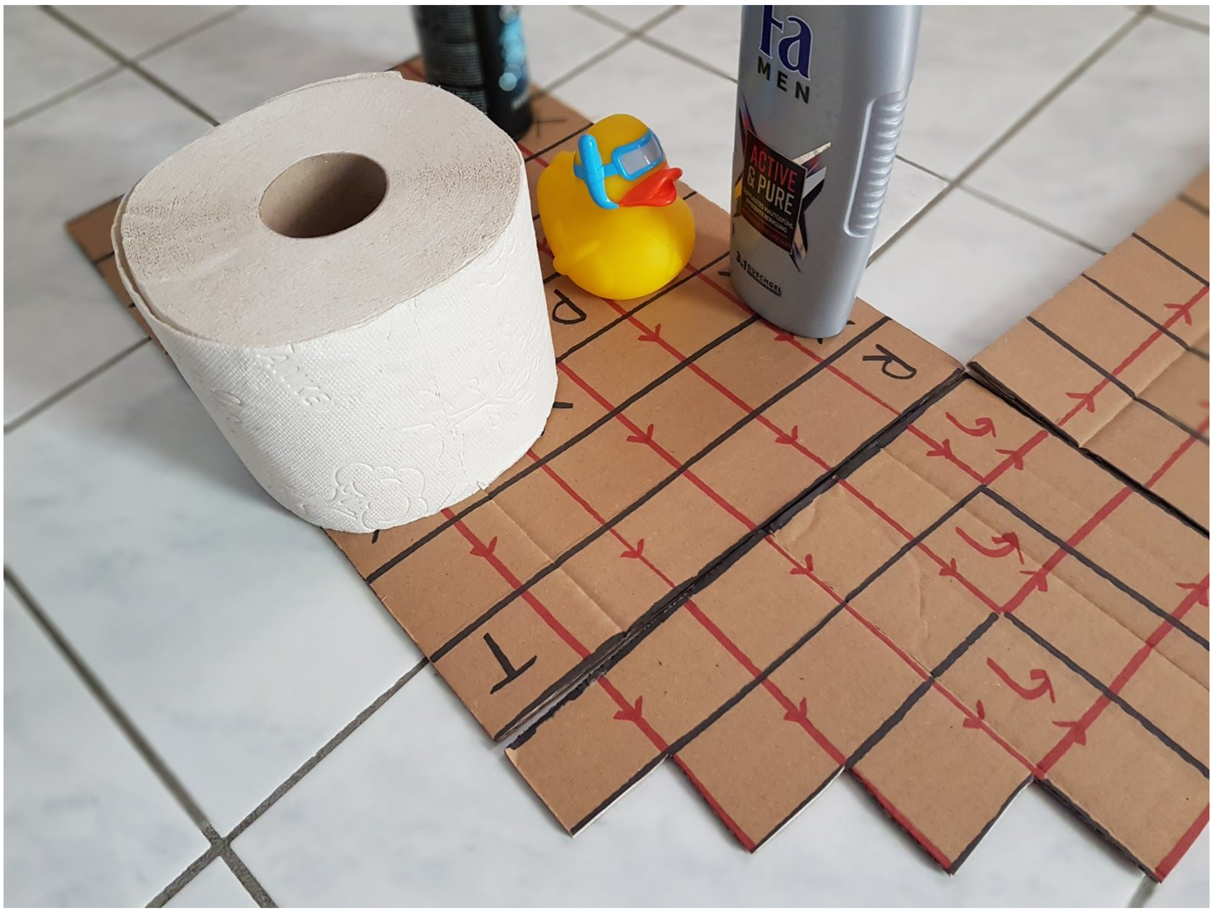
Who might want to play this game?
Everyone who needs a little distraction, while having to wait. People in public transit, children during breaks, etc.

What do they do in the game?
Trying to get as far as possible and gain a higher score.

What virtual world should the system simulate?
A world where an innocent duck escapes the burden of humanity by gliding on a soap to it’s freedom.

What criteria should be used to judge whether your design is a success or not?
If players want to play just one more round. Players don’t get bored after a few minutes.

2. Physical Prototype



2.1 Prototype idea development

The goal of the physical prototype was to show the core mechanic to the user, or viewer in this case. Since an actual physical playtest is not possible we thought about how to properly show the mechanics involved. The gameplay in itself is simple. You play a duck on a piece of soap. You slide, you dodge projectiles in a never ending fashion.

The original idea was to simply do what with the physical prototype and record it in a video to show it, expecting that this would provide sufficient insight on how the gameplay is supposed to feel like while keeping a small comedic touch.

However upon more thinking within the team and some research on physical prototypes we came to the conclusion that it is possible to cover the main aspects in a more gameplay related manner, while still keeping the core aspects intact.

So the idea we came up for the prototype that is going to be built is a turn based “racing-styled” game.

2.2 Prototype description

The prototype is made out of cardboard, because it is cheap and can be cut into same sized blocks. Apart from cardboard, a player piece, paper for notes, indicator tokens for water and some pencils are used in our prototype. Race track building blocks are simulated by having multiple cardboard pieces with track parts such as straight lines and curves.
The tracks are divided into 6 lanes which consist of steps. The player starts at a bottom field and needs to stay on track as long as possible. Water is indicated by blue tokens on the track. Obstacles and power ups are drawn onto the tracks in different colors. The track pieces can be combined in different ways and reused several times to simulate the infinite procedural track generation that will be included in the digital version.

2.3 Prototype Gameplay

Materials:

- Player piece
- Multiple set size track pieces
- Indicator tokens for wet floor
- Paper
- Pencils

Since the physical prototype is going to be quite different from the real game, some further explanation is required.
First of all the goal of the game remains basically the same. While in the real game the player has to survive as long as possible while also maintaining a high score. The real time component in this case is going to be replaced by a turn counter, while the high score will also be simply calculable by using for example a field crossed counter with some bonus points for clearing certain objectives.

2.3.1 The Track

Two track elements are always in play. Namely the one the player is on and the next one in movement direction. Whenever the player leaves a track part that one is removed from the track and added back to the track part pool. Then the player has to blindly draw the next part from the pool. The new part is added to the track so that the red arrows point in the same direction.

2.3.2 Turns

Instead of real time reactions, the main gameplay has also been changed to a turn based play style, with the player being able to make decisions every turn.

A turn consists of the following steps:

- Move forward according to your speed
- In the meantime apply all effects of fields you pass
- You may move one field left or right
- If no water field was passed this turn you lose one speed
- You may shoot water at a field
- Calculate your points for this round

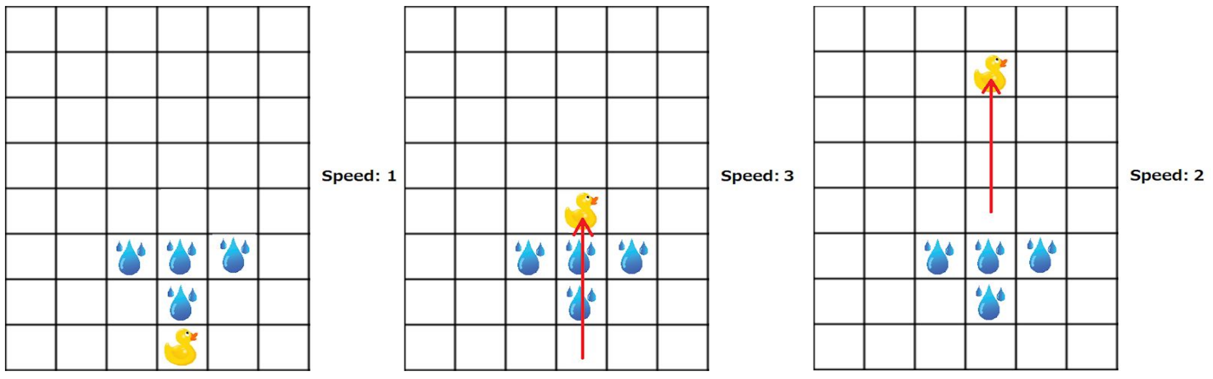
The game continues until the player collides with an obstacle, falls of the track or has a speed of zero.

2.3.3 Basic Field Types

There are several types of fields that have different effects while passing over them:

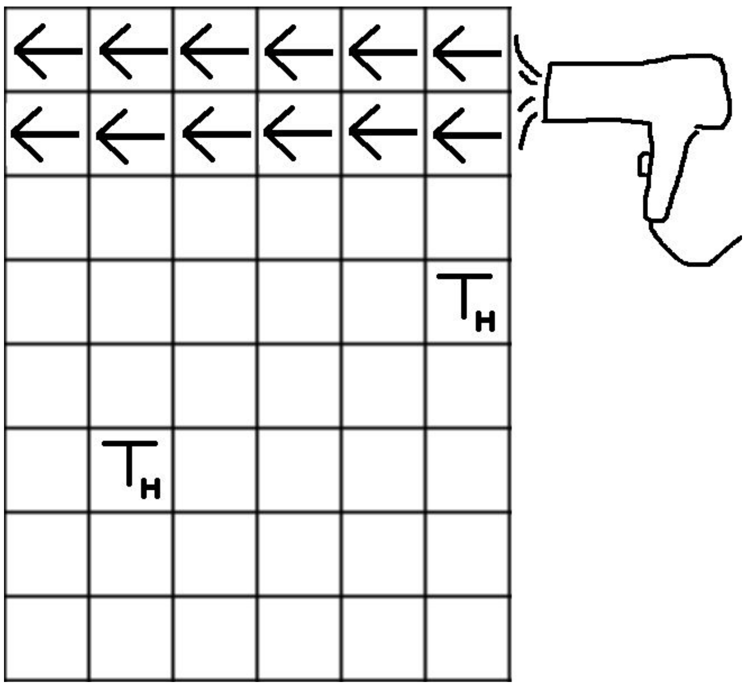
- X: Obstacles end the game immediately on collision.
- H: Holes end the game if you stop on them. Moving over them has no effect.
- W or fields marked with a water token: Add one speed point and let the player slide to the next field along direction arrows.
- R: Refill the water reservoir completely.
- T: Targets that grant a permanent multiplier if shot with water. They also count as obstacles.
- P: Powerups grant one additional field of movement to the sides for this turn.

The following image demonstrates the effect of the water fields:



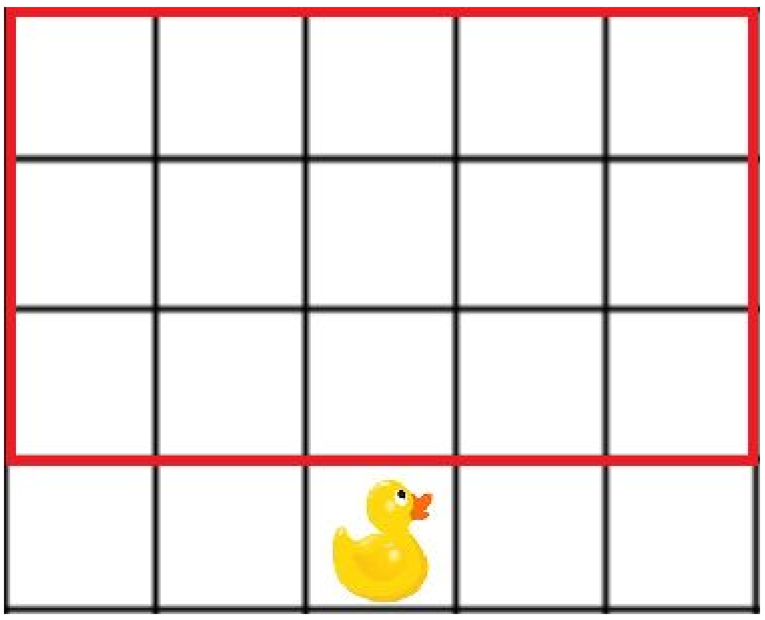
2.3.4 Special Obstacles

Special Obstacles are obstacles for which the player needs to do something other than dodge to overcome them. For this prototype we created one track part with a special obstacle, a hair dryer that pushes you off the track if you don't deactivate it first. To deactivate it you need to hit at least one of the special targets with water.



2.3.5 Water Shooting

The player has a reservoir that holds up to three charges of water. To shoot water the player needs at least one water charge in the reservoir. Water can be shot on any field in the five by three rectangle in front of the player.



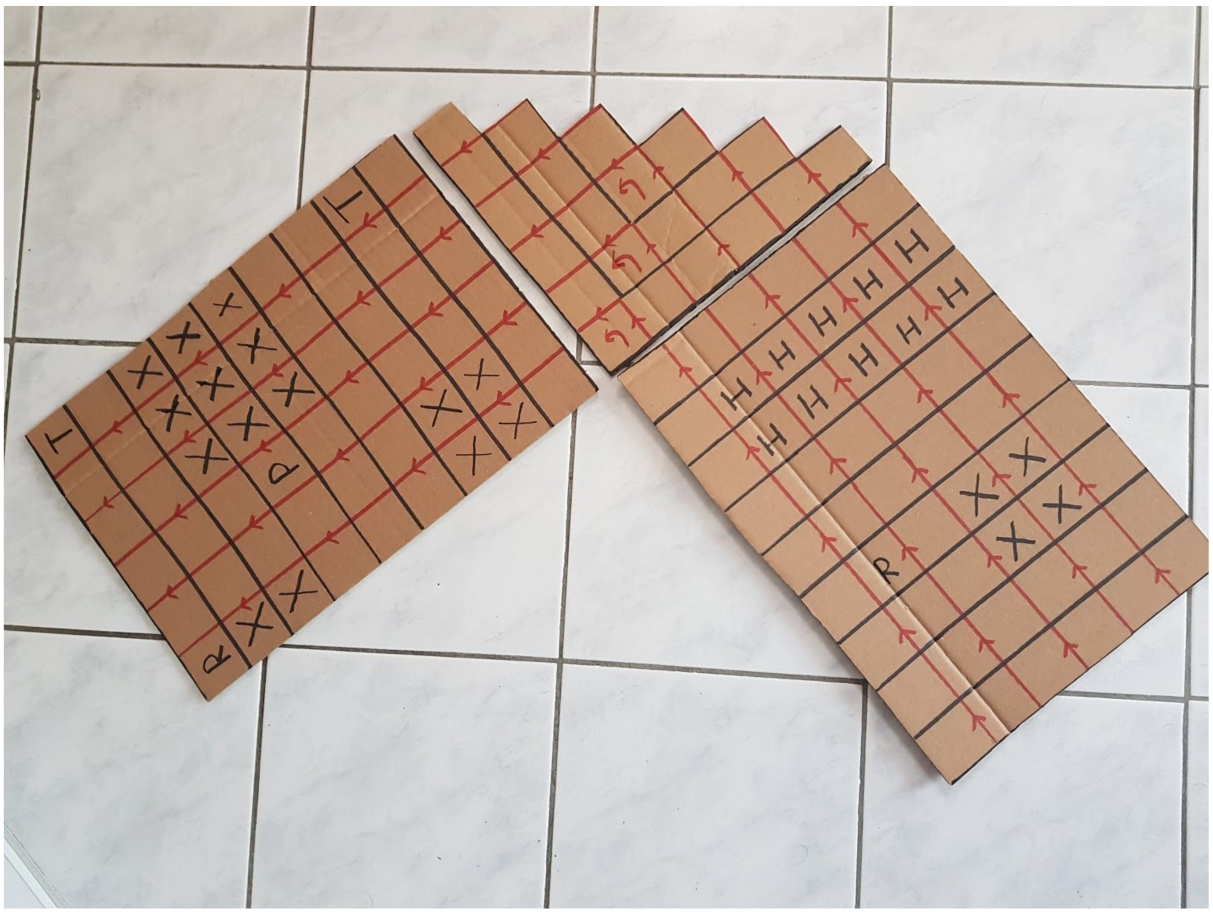
Water shot at a non empty field has no effect except if it’s a target field. Targets get activated if they are not already activated yet. Water shot at an empty field creates a puddle starting from the field.

2.3.6 Point Calculation

Points are calculated per turn and added together at the end of the game. Every turn the player gets points equal to their speed. This number is multiplied by the number of targets shot up to that point.

2.4 Prototype level

This is an example of how a track could look.



2.5 Next steps: from prototype to real game

Since our main game is more reaction and racing based it was impossible to truly recreate the main concepts into a board game. Instead we ended up going for a turn based game to keep the main mechanics the same. This allows for a close enough experience while still

For example, we have some obstacles and fields that get randomly placed by rolling a die. This is done to diminish the repetitiveness of placing the objects manually. Instead of repeating tracks one could also consider randomizing obstacles inside the randomized track pieces.

Via the prototype we tried balancing the secondary mechanics, which are not dependent on real time. We thought about how the speed-up works exactly, how the objects should interact with the player, how the high score is calculated, how to manage the water level and where to shoot it and how powerups should work and be interacted with conceptually.

2.6 What we learned from prototyping

Prototyping is a valuable method for testing different game ideas. It gives fast feedback, if a certain idea is valuable and should be considered further, following the principle of “fail faster”. This is not only valuable for testing game concepts, but also quite useful for testing certain mechanics of a game.

Nintendo has a popular Philosophy of Designing games, especially the Super Mario games, by having a Prototype with only the player (in unity often just a sphere). In this extremely limited environment they try to make the movement as fun as possible, which is the core mechanic of the game. After using this core mechanic, you can increase the scope with interesting additions, which change up or complement the core mechanic.

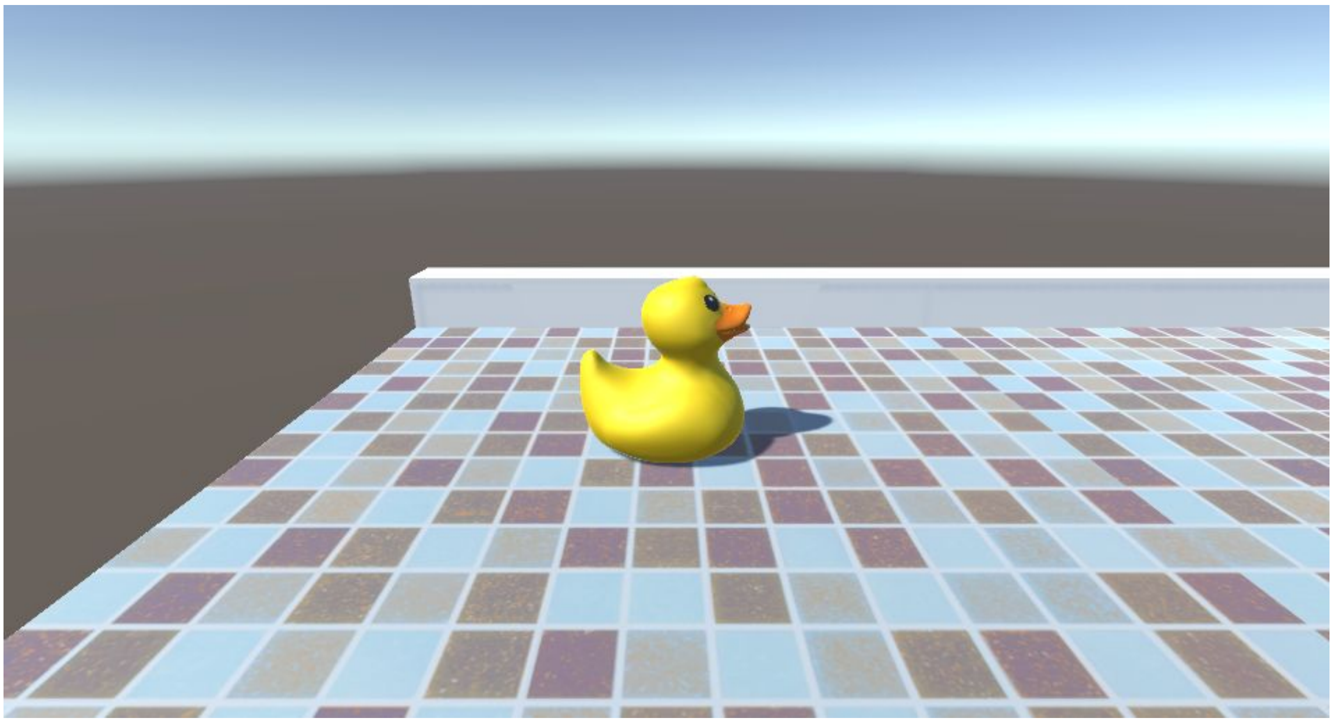
Most Prototypes are and should be throwaway. The reason for this is that you will not try to write good code, but fast code. And even more important, you should not already start with bad code, which destroys a codebase the longer you work on it without serious and time wasting refactoring.

A Physical Prototype is another kind of Prototype. This kind of prototype is fast to develop, especially if you are an unexperienced game developer. Another advantage is that you can’t adapt a bad code base, because there is none. A Physical Prototype is especially useful for any kind of strategic, tactic or puzzle game. Turn based games are the most fitting kind of game. For These Complex games, a Physical Prototype is often prefered, because coding takes much longer. Other fitting games are adventure games, story driven games and card games.

In other cases, like Action games, racing games, fps, Jump and runs and every other fast-pace game, Physical Prototypes often fall flat. The core mechanic of these kinds of games is almost always movement based, instead of methodical or combinatorial like in puzzle games. It’s extraordinarily difficult to test any kind of movement based mechanic with a physical prototype. These kinds of mechanics are best tested by playing around with them and getting a feel for the movement. This is impossible for physical prototypes.

Therefore a physical prototype has to be severely adapted to use and transform the secondary mechanics into a scheme, to make it testable in a more turn-based, methodical way. This is an interesting way to explore these secondary mechanics and can almost be developed in its own game. However the core mechanic will not be tested in a fitting manner to make it possible to make a decision, how the mechanic should be adapted or if you should drop it entirely.

It is also a bit more difficult to work on a prototype in collaboration. Usually you are dependent on other people, available material, time and location, which is not the case, if you use some collaborative workflow tool like git. This is especially difficult in a Lockdown phase, because most collaboration has to be at the same place at the same time.



3. Interim Report

3.1 Level Generation

The first idea for generating Levels was to have a purely random approach. The randomness for the interim results mainly focused on the obstacle placement. For this the collider of the ground was used to randomly place objects inside it with a margin on the border. The objects were placed completely random and therefore some problems appeared.

One problem was the placement of multiple objects. It has to be considered to place objects only on areas, where they don't interfere with others. This was not that big a problem, but still noteworthy.

More problems occured with the timing of the generation. It has to be considered, when the objects should be spawned exactly. A main focus for this is on the speed of the duck and the distance to the next point, which is visible. The best approach for solving this problem is to spawn objects outside the visible field at once for a whole segment. This should depend on when the player leaves the current segment and should trigger the object generation just outside of view. This obviously means segments have to be generated out of the track.

Another big problem is how curves should be integrated. With this the separation of the track is even more necessary. The curves can then simply be placed on the end of the straight tracks and generate a cohesive track in total.

The biggest problems appeared in regards to game fun and Game Design aspects. The problem with random placement is, that it is, well, random. With random placement it is not given that the track is fun or even possible. With a truly random track, there could be long empty areas and some impossibly packed ones. Additionally, even if the track generation would be possible, when you have segments it could still be that the connection with other segments generates an impossible to traverse path.

To make the game fun a priority, there shouldn't be any paths which make you lose instantly, because it's impossible to dodge the next obstacle. Additionally there should be a consistent level of challenge, with no long segments of boredom and no extreme difficulty spikes. The difficulty should ramp up over time through either obstacle placement or simply the character speed.

To make this possible, inspiration was drawn from Derek’s Yu philosophy of designing the critically acclaimed game Spelunky. Derek generates the level by having a collection of interesting challenging rooms, put together in a way to have at least one possible path from the start to goal. This makes it possible to have a random map which is possible to traverse and still has interesting gameplay.

We therefore plan for future track generation to build a collection of various small mini challenges, which have some free space at the start and beginning of each segment. This makes it possible to have an interesting, unique track with few impossible paths. The more track segments are collected, the more varied the final track should be. For alleviating the burden of manual track generation, track segments can also be mirrored.

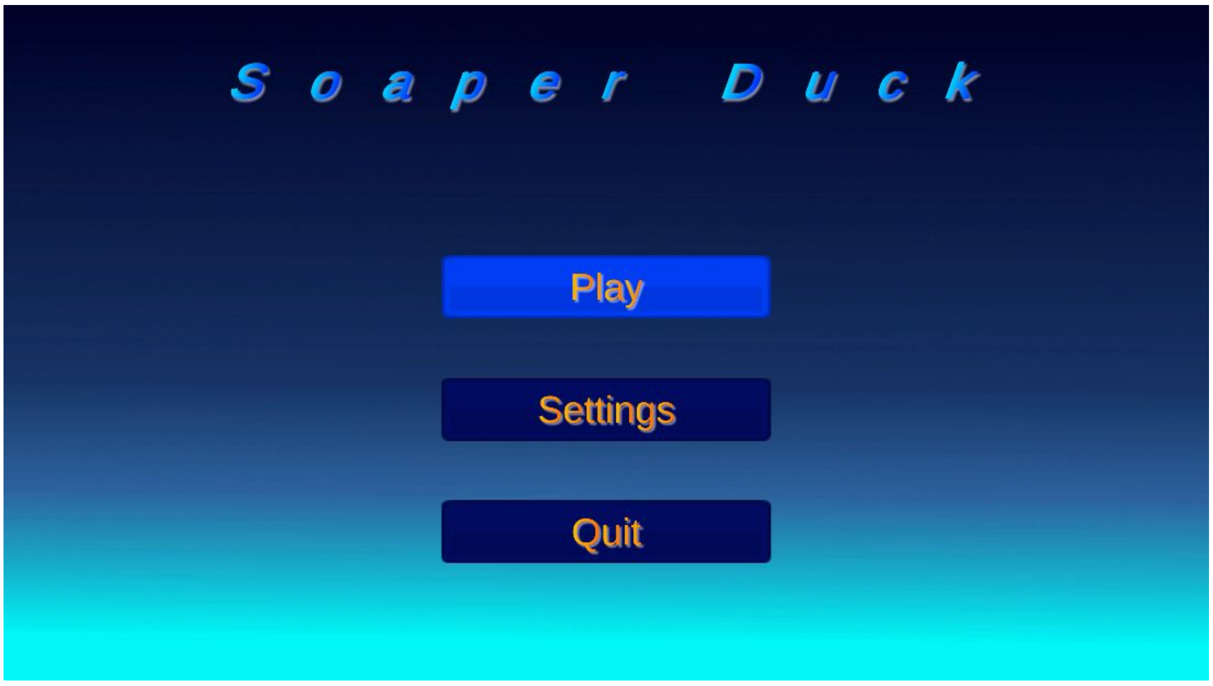
For the first results we however still use random generation and try to live with the problems mentioned before, until enough time is available for the new, pseudo random track generation.

The track generation itself (excluding obstacles) is quite simple yet effective on a technical view. Every track has a point set as the end of the segment, so for the endless generation of a given track, we first have to decide when to add a new track at all. That is decided with the distance of the player towards the current end of the track. If that distance passes a certain threshold, a new track segment is attached by spawning the new segment at the point where the previous segment ended. This supports different rotations of the segments, meaning that turns and changes in direction have been accounted for.

3.2 Player Controls

We concentrated on the basics of player movement for now. The player has the abilities to move left and right as well as to jump. Also the player moves forward automatically at a specific speed. We are currently testing different techniques for water shooting. It is important for the aiming to be quick so the player can react in time.

3.3 Menu and GUI



In our simple game menu the player has the options to start the game, set the settings and quit the game. The menu is optimized for 4k resolution, but works with every resolution. The text and button are getting scaled automatically so that the game could in the future also be ported to mobile or console without having to change the graphical user interface.

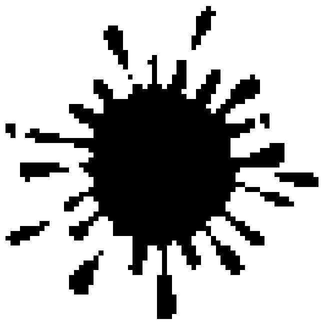
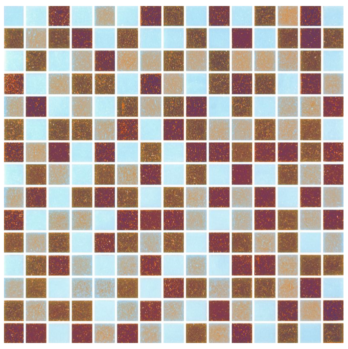
The title text uses Unity's new TextMeshPro feature that allows for more customization options like for example dynamic gradients, shadow effects, character spacing and mouse hover effects. The buttons are located in a panel that uses a vertical layout group to align the buttons on all resolutions.

The settings menu is invisible at the start and gets activated once the settings button is clicked, of course this will deactivate the main menu. Using a back button the main menu shows up again. The settings menu is currently empty, but we are planning to add options like for example sound volume later on.

The start buttons loads the game scene and the quit button ends the application when clicked, this is achieved via a simple C# script.

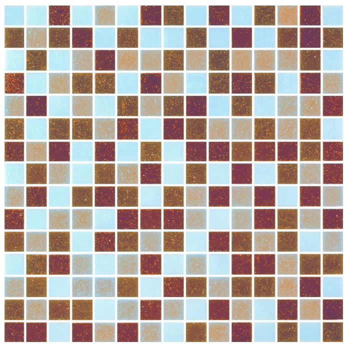
3.4 Splash and water shader

3.4.1 Current Approach

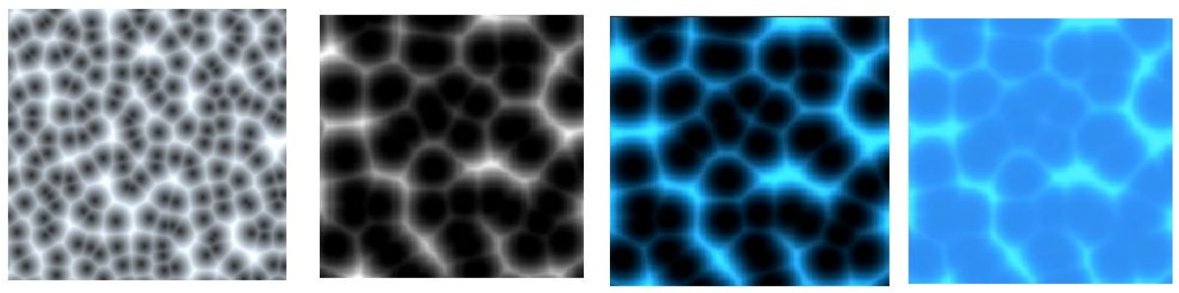



Our first approach to render water at the position where the projectile hits a surface is based on a shader and a splash texture mask. The splash texture is black and white, because the idea is to render this texture onto the ground texture and then use it as a mask for the water shader. Using a C# script and raycasting the position of the bullet hit point in world position as well as the UV coordinate on the ground texture is calculated. Using this UV coordinate as the middle point of the splash every pixel in the splash texture that is black gets rendered onto the ground texture and replaces the original pixel.

The result looks like this:

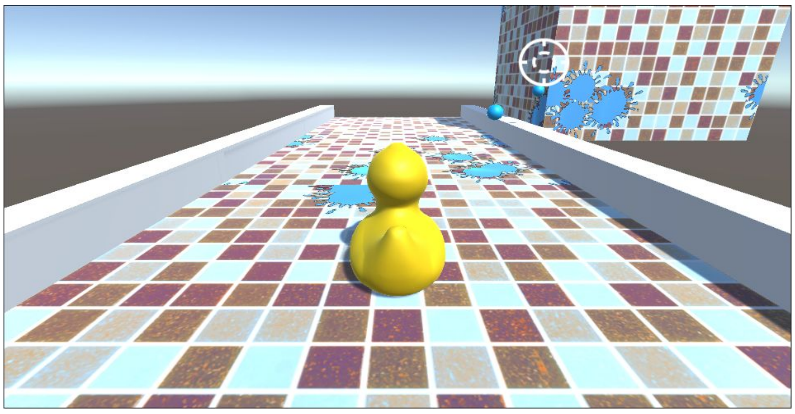


With this setup we created a very simple water shader using a Voronoi noise.
The voronoi noise has several cells and ripples, we only want to use the ripples and have the cells in one color. Therefore we increase the cell density and multiplay the white parts of the voronoi with a sky blue color. The blue base color is added to the black inner parts of the cell. Finally we added time to move the water in negative z-direction to make it look more realistic.



Sources:
Voronoi shader: https://www.ronja-tutorials.com/2018/09/29/voronoi-noise.html
Simple water shader: https://www.youtube.com/watch?v=Vg0L9aCRWPE
Original splash C# script (we edited it a lot):
https://sergeyshamov.wordpress.com/2018/05/22/paint-splat-on-objects-in-game/

Using a simple if-statement, that checks on runtime whether or not the current pixel on the ground texture is black, we render the water shader on top of the black splash parts which then makes for a decent looking water splash.

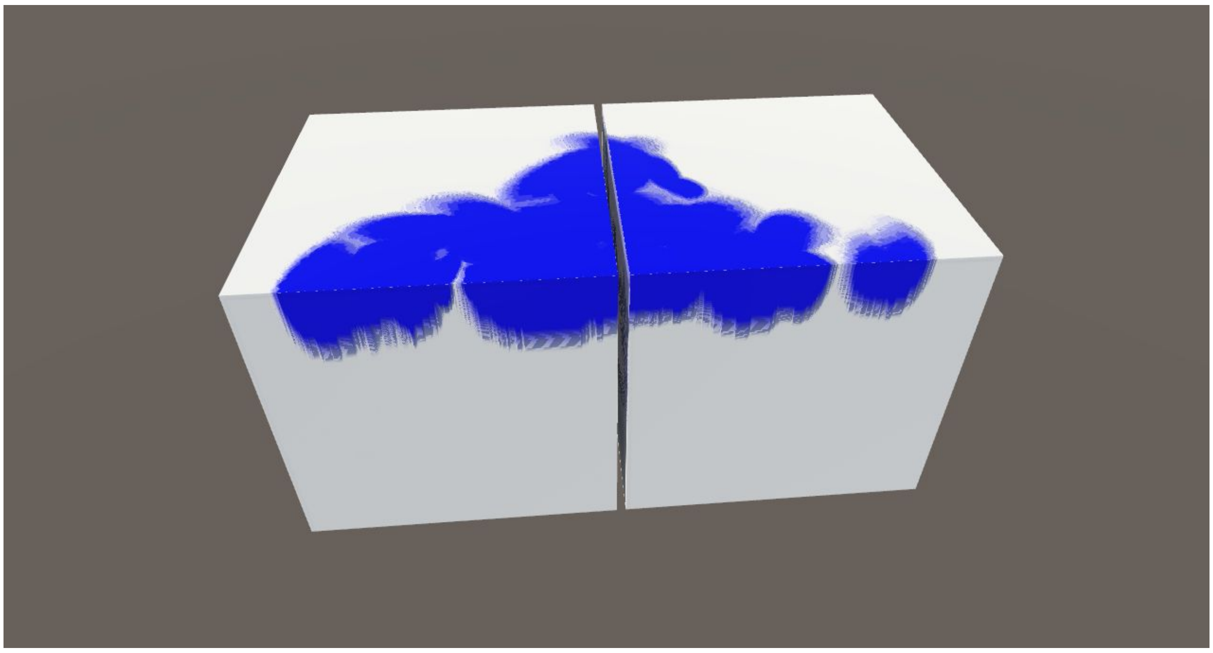


Problems with the current system:

- Splashes don't work on disjointed uv edges.
- If its UVs overlap, the splashes appear on multiple places on an object.
- Splash size is tied to object scale.
- Splashes don't wrap around edges.
- Raycast can only hit one object at a time, so splashes cant cover multiple objects that are close together.

3.4.2 Outlook and new Approach

Since our first idea to display water on the track has some major technical difficulties we are currently testing a different approach. If we render the splashes as decals instead of drawing directly on the texture we can circumvent many of the issues we had. We still need to figure out how to best create and organize the decals but first simple tests seem promising. We can hopefully also transfer the animation we created in the first approach to the decals.



3.5 Goal achievement summary

We have managed to implement all the points that we set for our functional minimum, as well as half of the features for our low target. With this we covered all the objectives we set ourselves for the interim results.

The chosen objectives result in a functional playable version of the game, where the player can dodge obstacles, shoot with a water gun and try to reach a high score as high as possible.