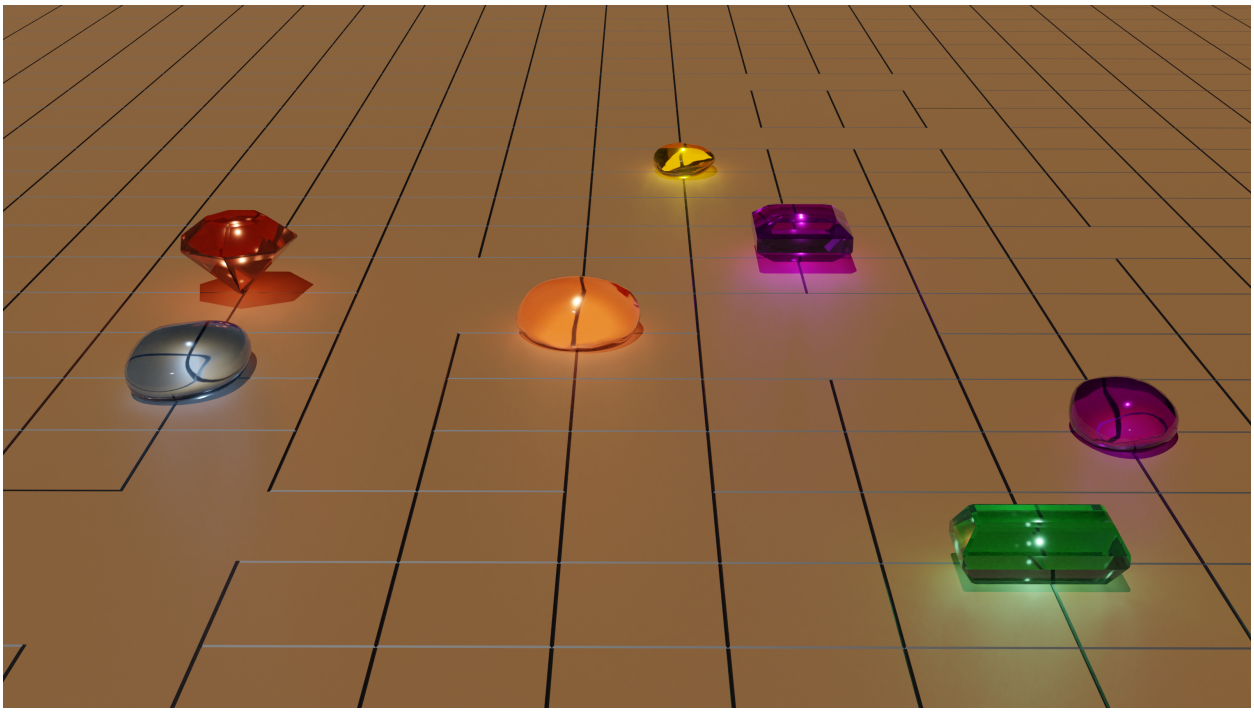
Alpha Release: *Gemji*

Team *DreiKopf*:

Felix Brendel
Jonas Helms
Van Minh Pham

June 2021



Contents

1 Game Logic Implementation

We previously had the game logic and the rendered scene implemented separately. The game logic at the time was running as a console application. While being 100% functional, this is far from the game we had envisioned. The scene in our engine that was rendered showed the board along with the gems but player interaction with the scene was missing entirely. Our main focus therefore became to combine both elements. The center aspect that is used for interaction is our raycaster that was implemented for the Interim demo. When the player clicks on a spot on the screen with the left mouse key, the raycaster determines the intersection of the resulting ray with the board (as described in the Interim demo). The 3D float coordinates of this intersection are then rounded to obtain the according 2D integer coordinates on the playing field. Since we know where our gems are and how big they are, we can determine whether or not the player clicked on a gem. To then move this gem the player has to hold the left mouse key and drag the gem to another location. On the key release the target coordinate on the playing field is determined with the raycaster as well. Should the target position not be viable- because it is more than one tile away from the original position, already has a gem or is out of bounds- the gem will instantly snap back to the original position. Having completely incorporated the game logic into our engine the game now performs the turn including the chain reaction after the player has moved a gem while also checking that his move is legal. Since the game logic was finished beforehand, this boiled down to a single function call.

2 Transition to 3D

As well as the game logic works in the engine, providing visuals to the player is vital to ensure a satisfying and enjoyable playing experience. At any time the player has to see what the current state of the board is, how moves turn out visually and where they are moving gems. In the following we elaborate on how we incorporated these visuals for *Gemji* into our own engine.

2.1 Gem Movement Along The Cursor

Gems move by holding down the mouse key and dragging the mouse across the screen which should be shown visually. To achieve this we have the mesh of the selected gem be the intersection of the mouse ray and the board while the left mouse key is being held down. It is worth noting that the origins of the gem meshes are at the very bottom center, so matching the mesh position with the mouse intersection caused no problems with height. When the player releases the mouse key, the gem mesh snaps to the selected coordinate on the board. A similar snap motion also occurs when the player moves a gem after clicking on it. Since the gem will have the origin of its mesh match the projected mouse position on the board, not clicking on the center of a gem will result in this snapping motion.

This would make sense if the player were to pick up a gem to then move it. But what we had in mind was for players to drag gems across the board. So in order to make this clear we added a 2D offset between the center of a gem mesh and the exact position the player clicked on given that this position belonged to a tile with this gem. With this addition gem meshes did not snap to the cursor anymore. However we noticed that the meshes were way off the cursor position at far edges of the screen. So specifically for the meshes we defined approximating spheres around them. The 2D offset thus became a 3D offset that was calculated from the mesh center and the intersection of the incoming mouse ray and the approximating sphere. This new offset has gem meshes stick closer to the mouse position even at far edges of the screen.

2.2 Animations

Animations are especially important to us, because our game is about understanding the interactions of the Gems between each other and their temporal order. Each gem effect has a different animation:

Blue gems Flicker before they teleport back to their original position

Red gems Explosively push other gems away. Movement starting fast, getting slower

Yellow gems Pull other gems towards them. Movement starting slow, getting faster

Purple gems Switches position with another gem, by letting both gems rotate around their average position by 180° around the up-axis.

We also use a simple parenting system where each object can have a parent object. This is used to animate multiple objects together, for example for the purple gem animation. Both gems are first parented to an empty object that lies in the middle of the to-be-switched objects. Then the orientation of the empty is animated to give the effect of both objects rotating around each other.

Additionally we use animations for simple camera movements at the start and end of levels as well as for fading tooltips in and out.

2.3 Scheduler

We chose not to apply animations in a simple frame-to-frame fashion. But rather give us higher level control over them. Therefore we put a lot of effort into the animation system, which is based on our work in last semesters practical. We wrote an animation system which we call the **scheduler**, which can animate any variable of any size in main memory. We use this to animate simple floating point values but also orientations and positions. To schedule an animation, you provide the scheduler with some information about it e.g when it should start, when it should be finished, what should be animated, what kind of interpolation should be used (linear, ease-in, ease-out, etc). Like this it is really easy to schedule animations to run at any given time in the future. When the player moves a Gem the whole turn outcome is computed in the same frame, and alongside of computing the chain reactions, their Gem movement animations are scheduled.

Additionally to executing animations, the scheduler can also be used to schedule **actions**. An action consists of a function pointer and optionally a set of parameters that the function pointer will be called with – essentially emulating closures[1]. We use actions when we know that after some time, *something* should happen. For example if the player performs the last move, that will set the board into a winning position, we first want the whole animation to play out and only then switch to the next level. So as the player performs the winning move and the board is simulated, and all the chain reaction animations are scheduled, we do a check if the board is in a winning position, and if it is, we schedule the action that loads the next level. The playing of sound effects is also powered by the scheduler to be able to play the correct sound effect at the correct time.

[1] C++ lambdas were not used, to be able to store actions packed next to each other in a flat array. This is not possible with C++ lambdas, which have different sizes, depending on their parameters. A higher level abstraction such as `std::function` would have to be used. We opted to not do this, as `std::function` can perform heap allocations on its own.

2.4 Grid Creation

Originally grids were separate meshes that were built in Blender. As grid sizes vary from level to level, having to create a separate mesh for different dimensions is way too tedious. Therefore we have shifted grid creation from blender to our engine. The necessary vertex and index buffers are filled using a function that takes the grid dimensions as input. So when we create a new level, all we have to do for the grid mesh is to call the function with the dimensions of the playing field as inputs.

2.5 Tooltips & Textures

One aspect that we deem very important for puzzle games like *Gemji* is the explanation of the game mechanics without resorting to pure text which is boring and tedious to read. This is even more important as the next milestone is the playtesting stage which will also be the first time that people not on the development team will be playing our game. For this reason one of the goals of the alpha stage was to design and implement tooltips that explain the behavior of the different gems.

2.5.1 Design of the Tooltips

For now we implemented one tooltip for each of the existing gems in the game. For the design of the tooltips we decided to show a before and after screenshot of the effects of the respective gem in an example game state. Additionally we added a title to each of the tooltips that is also color coded. Although the title does not deliver crucial information to the players it does help in terms of making it clear which gem is the focus of each tooltip and will help us for the questionnaire and talking to our testers in the playtesting stage.

2.5.2 Implementation of tooltips

For the implementation of the tooltips we used the ImGUI library which we have integrated into our rendering pipeline. The ImGUI provides easy window creation for all UI elements that we want to use and will be also be helpful when we want to create an Options menu to control elements like the sound volume of the music/effects and rendering options. The tooltips are shown in the bottom right corner of the screen when the player hovers over a gem using the glfw mouse position callback. When the player stops hovering over any gem the tooltip will fade out after 3 seconds. Additionally the tooltip will stay active when the player holds the gem actively with the mouse before making a new move. The animations for the tooltip fading are implemented by our scheduler animation system.

2.5.3 Finish tiles

The goal for a round of *Gemji* is to move gems to their respectively colored finish tile. Up until this milestone we did not have a representation of the finish tiles and even the command-line version did not mark the finish on the ASCII boards. This crucial aspect was very important to finish otherwise it would be very difficult to explain to the play testers what the actual goal of *Gemji* is. Additionally the design of the finish tile textures influences the perceived game setting for the players. For now the finish textures were designed as magical circles but could be subject to change in the future. Furthermore the game objects will be automatically created and placed to the correct locations depending on the information in the "playing_field" struct when a level is loaded.

3 Sound Effects & Music

Puzzle Games usually do not have a good way to immerse the player in a narrative. This is especially true for *Gemji* as it is a very abstract puzzle game. Immersing the player in the game world has to be achieved by other elements such as SFX, music and particle effects.

3.1 Sound effects

Finding fitting sound effects for *Gemji* was definitely a challenge not only because of the search process but also due to the relation to the "game world". Just as with the finish tiles we decided to go into the direction of mystical/magical SFX that relate to the effect of the respective gem. In the following section we describe the SFX as they relate to the gem effects:

Red gems Pushing - Magical shockwave

Yellow gems Pulling - Slow starting but abrupt ending wooshing

Blue gems Back teleportation - Channelled teleport sound

Purple gems Swapping gem - Magical rotation

3.2 Music

Just as the SFX the music has a major role in the players immersion of the world. For now we wrote a simple asian-style mandoline melody for *Gemji* as we feel like we could go into asian-influenced theme. This inspiration also came from the game board of the physical prototype where we used a Go-board which we now also use as representation of our board in game.

4 First Campaign Levels

From last semesters project we gathered additional experience how we want to setup the level structure. The level structure is the biggest influence on the learnability of our game. The levels have to be structured in such a way that they not only introduce the basic game mechanics in a sensible matter that allow for good understandability and a smooth learning curve but also slowly teach what we call emergent effects. Emergent effects are the resulting game patterns that depend on the interaction of the different game mechanics working in combination. This mainly includes the different gem effects interacting with each other but for example also in which order the effects are triggered. The level structure and the learning curve is also something that we want to closely analyze for the playtest.