Stonehenge

Marlene Mayr
S1910629016

Abstract

Stonehenge is an augmented reality game challenging players to recreate structures with building bricks. The project was a collaboration with Eric Thalhammer to create a digital version of a board game for smartphones.

1 Introduction

Stonehenge is based on the board game classic *Make 'N' Break* by Ravensburger [1]. The concept of this game is simple: the players try to reconstruct building instructions with wooden blocks and get more points the more structures they successfully build during a given amount of time. It is a mixture of dexterity, remaining calm during the stress and taking on the challenge. Our aim was to develop a similar application that can be played on a smartphone, which provides equally as much fun and makes use of the benefits of three-dimensional space. In the end an interactive, polished way to have fun in AR should be presented.

2 Game Design

The main concept follows the original gameplay with a few tweaks. Players have to build structures with rectangular blocks, the *bricks*. The building instructions are shown as miniatures of what has to be built, but the structures are not limited to planar building like in the original. Instead, three dimensional structures can be built as well, but the instructions stick to 90 degree, block-like alignments. The score is not directly evaluated by the amount of structures that are successfully built but has been adjusted to represent the difficulty level and, therefore, is multiplied by the amount of bricks necessary for the structure. Similarly to arcade games, the players get a small time bonus upon finishing a structure which gives them the chance to earn even more points through successful building. The achieved score is not only saved during the round but allows for competition through an integrated online high score board. Furthermore, upgrades to the project like multiplayer support and several accessibility features already lie ahead.

Internally, the building instructions are called *recipes* and the building panel that checks for correctness is called the *mold*. We figured that this comparison fits the needs and functionality of said components.

3 Implementation

During the implementation, we developed multiple features necessary, always trying to follow good practices and producing a well-structured project architecture.

- The project has been set up with Vuforia in Unity to quickly get started. This also allowed for easy testing during the development phase directly in the Editor view with any webcam.
- Recipes are saved as `Scriptable Objects`, special data containers provided by Unity which automatically serialize their content and can easily be referenced in objects in the scene.

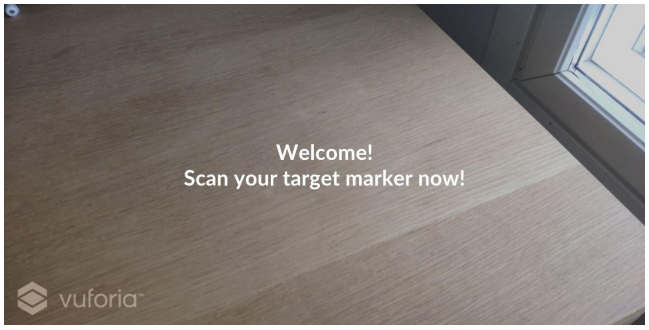

Figure 1: The screen appearing at startup as long as the tracker has not been detected.

Figure 2: The main menu already previews the playing area in the background and information on the mechanics of the game.

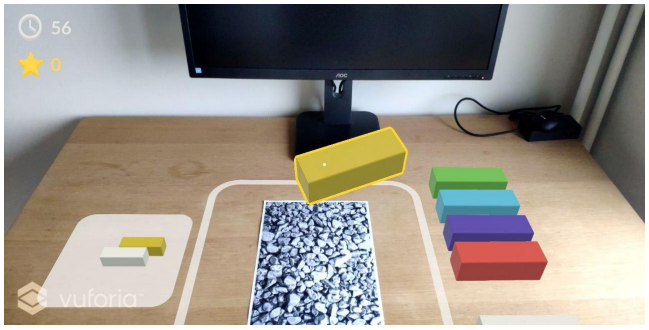

Figure 3: This screenshot shows the recipe on the left and a brick being picked up. However, the playing area can be rotated by moving the tracker.

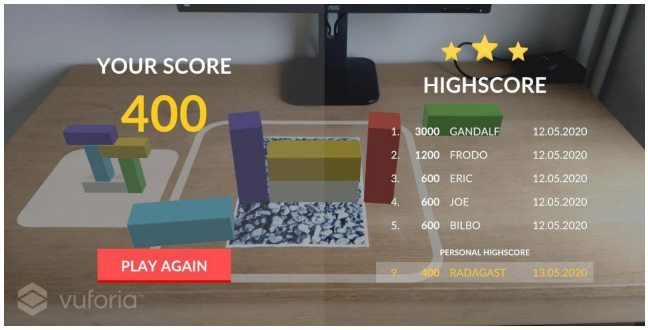

Figure 4: After the time runs out, the application presents the personal score and an online high score board with the reached position.

The content is saved as a collection of bricks which in turn contain the information for the correct identifier (which is mapped to the color of the bricks) and the positioning. Multiple recipes are referenced in the *cookbook*, a GameObject which handles the retrieval of the next recipe that has to be built. The selection of the recipes follows the order they are saved in the cookbook but can be randomized as well.

- It is important to note that the recipes only save the data but no objects. Therefore, when the recipe is loaded, a visualizer takes all the information and places bricks according to the data. These bricks have the same material as the movable ones but are instantiated without any colliders or physics body since they need not comply to any physic input. Additionally, the visualization is scaled to resemble the miniature, instructional feeling.

- The game flow is handled by a custom state machine that follows the state pattern of software design. It switches through the various states in the game which have their own implementation of handling events like the loss of tracking and menu inputs. This allows the application to guide the user through the application with the help of on-screen menus and HUD-texts since AR applications are not as wide-spread and need more explanation about the interactions than regular mobile games. The application starts in a state which tells the user to point the device at the tracker because the game would not be able to start otherwise. The HUD is represented in Fig. 1. Afterwards, a menu like Fig. 2 appears with brief instructions on how to play the game. This is the point where further adjustments could be made if settings are necessary in future upgrades of the project. When the user clicks on `"Start game"` the menu disappears and while the game area is already visible, a short countdown gives the user some time to get accustomed to the surrounding. Then the main game loop handles all interactions, the checking of the built recipes, score, timer and providing new recipes on success. Picking up bricks is shown in Fig. 3. After the time runs out, the user is redirected to a score panel. When entering a name, the score will be uploaded to the online high score board displayed in Fig. 4. The user can then take up this new motivation to get better and click `"Play again"`. Upon clicking, the state machine activates the game state again.

- The user interface that helps guide the player through this process follows a simple flat design provided by my team partner. This style accentuates the difference between the augmented world which is combined with the real world and the screen space menus. The UI scales with various screen sizes and especially aspect ratios to always fit all important information on the screen. A careful approach and some testing was needed because modern smartphones come in many sizes and ratios.

- To match the bricks with the real world we tried to keep the material and lighting as close as possible to painted wooden blocks in an indoor environment. The custom material was adjusted to match the original game and transfer a similar playful look and feel. The scene is lit by a directional light and an indoor image sphere as a skybox for environmental lighting.

- As already mentioned before, the bricks always have to be aligned in 90 degree angles to each other. We developed a system that would allow for easier verification of the structures but also helps the user control the bricks. It can be rather tedious to exactly place objects in augmented reality due to spatial recognition and the movement of the device. We did not want this to become an issue or diminish fun because of difficult brick handling. Therefore, a voxel space was introduced. Voxels represent data points in a three-dimensional space similar to coordinates but with a unified sizing and the representation of volume. In our case, every point in the 3D scene can be mapped to a voxel to get the information in which voxel grid place it is positioned. The scaling of this voxel space is adjusted to fit our playing area so that a brick always has the size $1 \times 1 \times 3$ voxels. The brick itself contains three anchor points, each positioned at the center of one of these voxels. These anchors are used to match the brick to its position. With this technique, the matching of brick is rotation independent. No matter if it is rotated by 180 degrees, slightly tilted or even rotated lengthwise, the brick will always return the same three voxels that it occupies as its position values. These are the values stored in the recipe, which is why there also exists a conversion from voxel space back to world space to be able to place the bricks. Verification of recipes is eased because only the voxel values have to be checked and not all possible position and rotation values within certain thresholds. Interaction with the game is facilitated because we snap the brick to the voxel grid when dropping. This way, when the user drops two bricks just near to each other and slightly tilted, they will end up exactly aligned. The snapping is interpolated over a short time to not be visually distracting.

- Although the voxel space brings a big advantage, the recipe-mold checking still has to take into account multiple potential problems. First of all, it is not executed every frame but interval-based to reduce computing times. Additionally, it is redundant to check all the bricks when there already is a missing brick compared to the recipe. Similarly, when checking a brick with a specific identifier and at least one voxel does not match, it means that the whole brick does not match the comparative one in the recipe. In these cases the methods immediately return, which again saves computing power. To assure that a structure can be built anywhere on the mold area, the brick data is not saved in the world's voxel space but relative to a reference brick. Which means that the brick with identifier 0 is always part of the recipe and all the voxel coordinates are saved and matched relatively to its position. Due to this implementation, the reference brick always has the voxel values $(-1,0,1)$ in the direction of its major axis.

4 Additional Implementation of Development Tools

Many of the aforementioned features are hard to imagine without visualization or help, even as the developer. For this purpose, several development tools were implemented that help building the application and creating content.

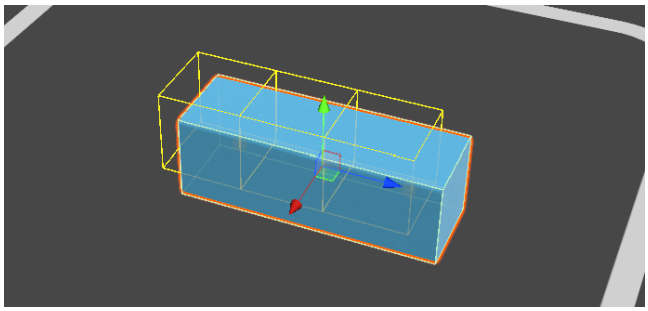

Figure 5: The brick gizmo shows the voxels that a brick occupies and tells the developer where the brick will snap to.

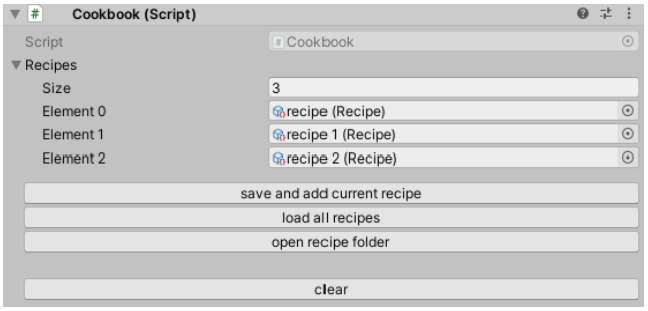

Figure 6: The custom cookbook editor provides the functionality to save and apply recipes.

- The state machine exposes two fields in Unity's Inspector: the initial state and the current state. This way, during testing, the developers can enter the game at any desired point and do not always have to run through the whole application flow to get where they want. For testing the state handling, the current state is displayed to check if every transition is executed correctly.
- Bricks in the scene view are shown with custom gizmos as in Fig. 5. The three anchors inside of the brick are represented with spheres and the voxels it occupies are drawn as wired cubes on the outside. This helps a lot to see where the brick is effectively positioned and what voxel values will be saved.
- Additional tools have been added to align the bricks to these voxels during the editing phase, which snaps either the selected or all the bricks in the scene to its nearest voxel positions and rotations. These tools can be accessed via the `/Tools` menu or via the shortcuts `Alt + A` and `Alt + s`.
- With the help of this visualization and alignment, new recipes can easily be built. Developers can place the bricks with correct alignment and see exactly what the recipe will look like. The most complex tool is the one that saves this recipe as a scriptable object. It can be accessed by clicking on a cookbook in the hierarchy. These have custom editors which add a few buttons to the inspector shown in Fig. 6. The most important of which is the `"Save and add current recipe"` button, which performs multiple actions in the background. It gathers the information of all the currently places bricks in the scene and saves the values to a new recipe instance. This will be saved as a scriptable object into the specified `/Recipes` folder. Furthermore, the cookbook's collection of recipes will be expanded by resizing and adding the new recipe, but also applying this change to the cookbook instance. This also works in the scene view, even though the cookbook is a prefab instance. With this tool, the content creator can easily save and add newly built recipes to the game with one click.

5 Issues

Several issues related with Unity occurred during the implementation.

- We ran into serialization problems when creating the recipes which led to recipes with missing or faulty data. After some research we found out exactly what unity can and can not serialize and realized that some changes had to be made in the custom brick and voxel classes to keep the data. All classes must have the `[System.Serializable]` attribute and all fields must either be public or again have an attribute called `[SerializeField]`. Static fields can not be serialized at all.
- Early on in the project we encountered physics issues in Unity when trying to stack multiple RigidBody objects on top of each other. The colliders start to overlap when there is too much weight applied, even when using regular box colliders. After some research we found out that this is a common problem and there are some ways to improve it but no fix. This was another reason why we implemented the voxel snapping.
- The determination of the correct rotation and position when snapping to the voxel grid led to some errors where the brick turns more than 90 degrees. This was changed to always choosing the nearest value to straighten the brick and taking both positive and negative directions of the axes into account.
- A related issue occurred in the rotation when snapping, which has a different underlying problem. All the rotations can be exactly one direction or mirrored, except for the lengthwise rotation. The brick can be placed in four possible alignments along its major axis. So, this had to be handled specifically.

6 Possible Extensions

- The recipes can be organized by difficulty or any other attribute. This can easily be done by assigning them to various cookbooks and loading only the necessary collection. Since the recipes can be freely assigned, this can be done without specifying the attributes in code.
- Checking a recipe is currently only supported in one direction, namely the same as it is displayed by the visualizer. This can be improved by matching multiple constellations of the voxels because the mold provides different relative voxel data when the whole structure is rotated.
- The application can be extended with a multiplayer support via a local network. The gameplay implementation, however, is still open for discussion. The players either build the same structures in competition but everyone on their own playing area or the players get different recipes that share bricks.
- Because we discovered that playing around with the bricks independent of competition is fun as well, we would like to add an edit mode in which the user can build recipes and even save, share or submit them. This would also help the content creators because the community can help with recipe creation.
- The whole application currently is scaled to fit on a table but it might be fun to play a bigger version on the floor as well. Although the scaling can easily be adjusted to quickly implement this feature, this would require settings with a toggle for scaling and, depending on the continuous tracking quality, also a different tracking image.

During the implementation we also figured out that some of the adjustable modules might be beneficial for specific people. For this reason, there are a few accessibility features that we want to add and evaluate:

- People with less dexterity or problems in spatial awareness might benefit from self-aligning bricks. Currently only the dropped brick snaps to the voxel grid. When moving the device or other bricks near placed ones, however, the bricks move freely according to physics. With this feature the player could still knock down the whole structure but does not have to be as careful when placing bricks on top of or next to each other. When dropping a brick, all placed bricks will re-align themselves automatically.

- Because we both know of people with some kind of color blindness we want to address this issue as well. I even know of one case where the player would repeatedly lose in the original board game because of this problem. The current implementation is already based on color palettes as predefined data containers for the colors of the bricks, but it only contains the predefined colors. Additional palettes can be set up for various grades of color blindness and similar visual impairments and then made exchangeable through a settings menu.

7 Conclusion

After multiple tests on our smartphones we agree that the game quickly sparks the competition within oneself to get better. Steering the bricks is fun, regardless of the competition and we are waiting for more recipes to test our skills.

References

[1] A. Lawson and J. Lawson. Make 'n' break. Board Game.