Project System Development SS2021

Structure

1. Overview
   - Starting situation
   - Main goals
   - Roles and groups
2. Workflow
   - Scrum
   - Communication
   - Git
3. Teams
   - Tech
   - Game-Logic
   - Simulator
   - Tracking
   - Effects
   - Display
4. Showcase
5. Performance
   - Problems and Solutions
   - Recommendations
6. Conclusion

Overview

Starting Situation

- Documentation was all over the place, difficult to understand
- Most Billiard rules were in place but a lot was missing or seemed wrong
- Project was split up in many different parts
- UI was implemented in a way where extending it was difficult and getting it displayed on the real Billiard Table was impossible

Main Goals

- Reworking UI to be easily usable, extendable and easy to display on the real Billiard Table
- Simplify Project
- Improve and overhaul the documentation
- Get the Simulator to a state where playing Billiard is possible
- Start implementing Image based Tracking

Roles

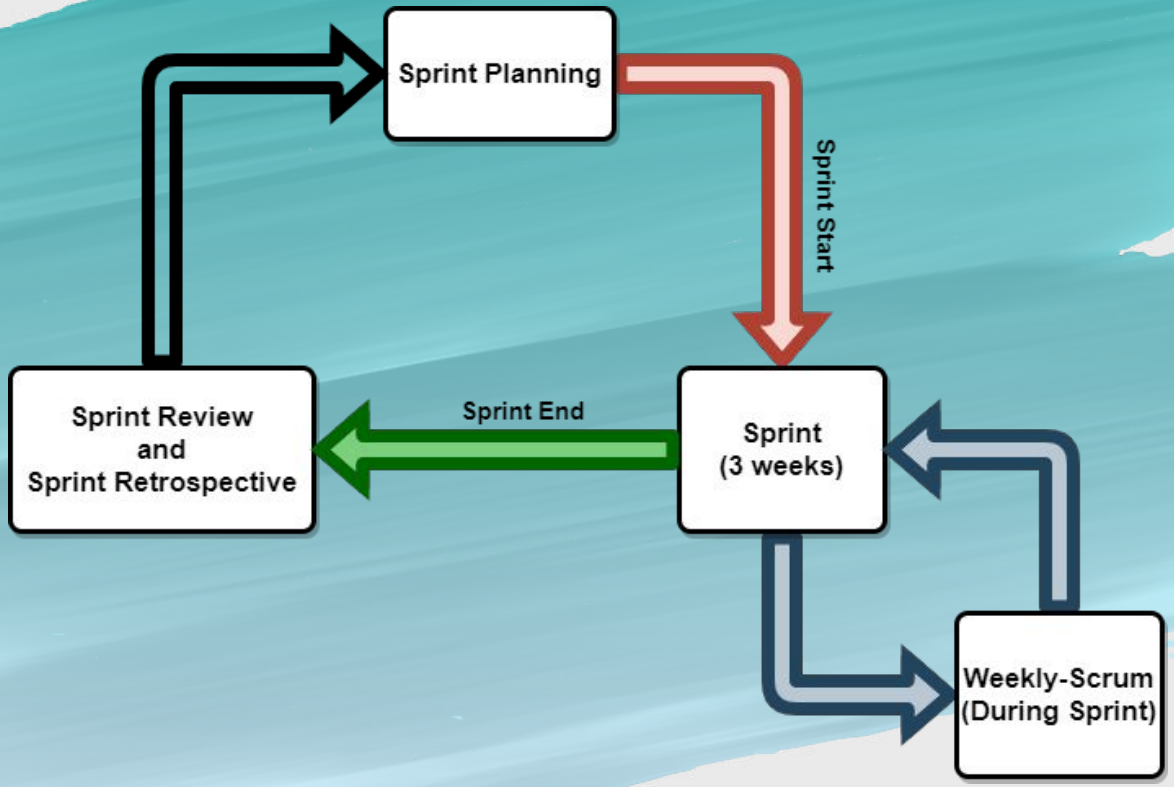
Position	Names
Product Owner	Felix Haßler
Head-of-Tech	Marcel Haidacher Felix Haßler
Scrum-Master	Simon Wenk
Gitlab-Maintainer	Lukas Frischkorn

Groups

Group	Names
Game-Logic	Dominik Deisenroth Julia Schmeil
Tracking	Kyrill Abrams Fabian Zickwolff Jan Schnurr Johannes Sauer
Simulator	Michael Reis Niroj Sedhai
Effects	Jan Blanquett Jaqueline Haberl Michelle Ehrhard
Display	Simon Wenk Lukas Frischkorn

Workflow

Scrum-Workflow



Communication

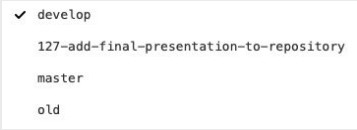
- Main platform for group-communication was Discord server from predecessors
- We held frequent meetings outside of Sprint Review and Retrospective
- Comments on GitLab-issues to share information

GIT(Lab)
Workflow

GitLab Setup

- Labels

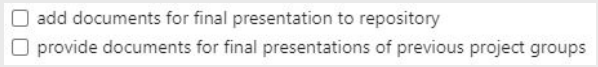
- Branches



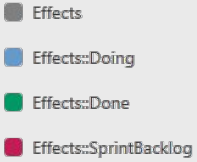
- Milestones (Sprints)

Create Task (Issue) in Gitlab

1. Create task with sub tasks



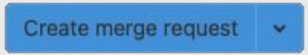
2. Assign label {TEAM}::SprintBacklog



3. Product Owner decides what Task should be done
   - Add the correct Sprint (Milestone) Tag

Working on a Task

1. Create merge request from selected task (issue)



2. Checkout the created branch
(auto generated by merge request)

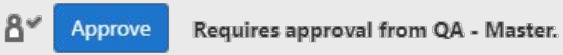
3. Commit changes

4. Push to server

Finish a Task

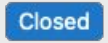
1. Mark merge request as ready

2. Get Approval from Maintainer



3. Resolve conflicts

4. Merge into develop

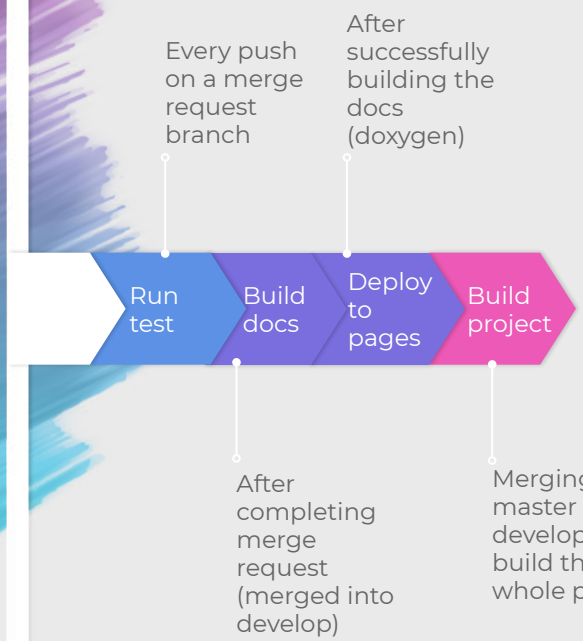
5. Issue gets closed 

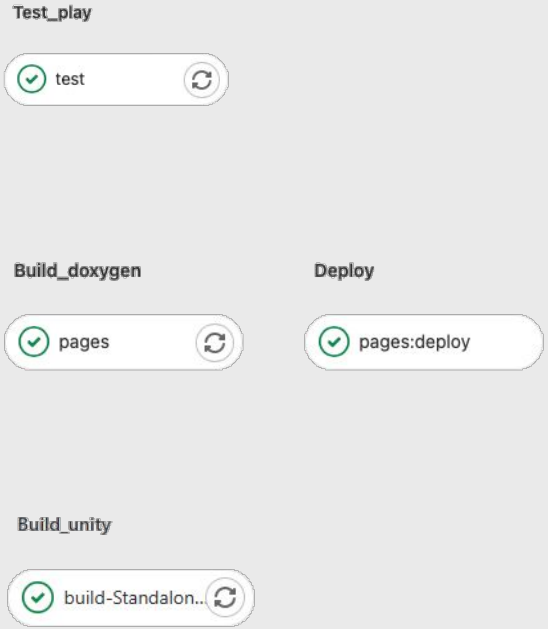
Tech

Goals

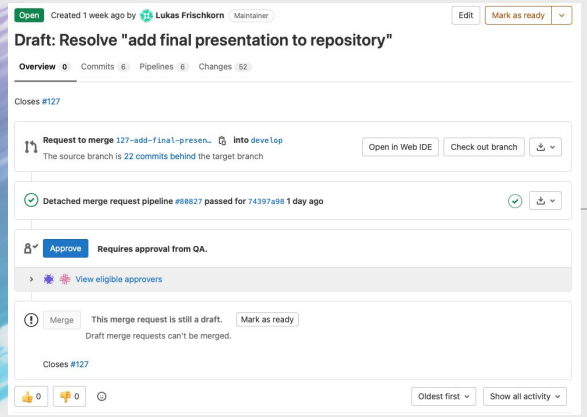
- Helping People
- Pipelining in Gitlab
- Documentation
- Tests

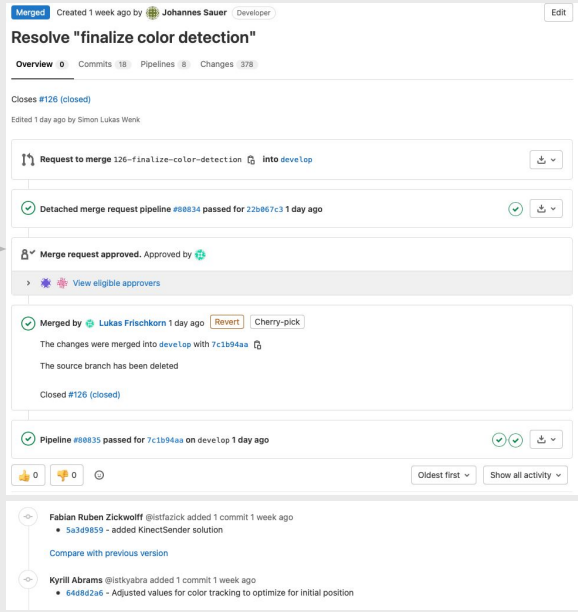
Gitlab Pipeline





Gitlab Pipeline





Ideas for successors

- Continue on improving the testing
  - Display test results in Gitlab
- Communicate with University about pipeline timeouts
  - Building the whole project sometimes fails when the Unity Docker Image is not cached anymore and needs to be downloaded (timeout)
- Add whole documentation in MD to doxygen

Game-Logic

Goals

- Restructure and comment the code
- Complete rules for 8 Ball
- Implement basic trickshot mode

Changes in the code

- Removed the PlayerTeamUtil class
- Removed the BallTypeUtil class
- Added a Ball and BallHandler class
- Added a list containing the remaining balls of a party to the player class
- Changed the EndGame consequence
- Removed separate Unity project
- Changed UI access to be via the Display-component
- Improvements to efficiency
- Commented every non-Getter/Setter method

8 Ball foul rules

New feature

- Pocketing the cue ball
- Hitting a ball that is not of the players assigned group first
- Pocket opponents ball
- Player hits correct ball but does not pocket or hit cushion
- Cue ball does not hit any ball
- Breakshot: player does not pocket any ball except black and cue ball or shoots at least four balls, excluding the cue ball into a cushion

8 Ball winning shots

New feature

- Player pockets black after pocketing every ball of his assigned group in the pocket opposite of where they pocketed the last of their assigned balls
- Opponent pockets black in a wrong pocket
- Opponent pockets black ball before permitted
- Opponent pockets black together with cue ball


TRICKSHOT MODE
HITS AVAILABLE:
NUMBER OF BALLS:
SELECT POCKET:
(YOU CAN ALSO CLICK THE POCKET)
UPPER LEFT
PLAY

Trickshot mode

- Game is won, if...
  - the chosen number of balls are pocketed correctly within the given number of turns
- Game is lost, if...
  - all available hits are used up
  - a ball was pocketed in wrong pocket

Ideas for successors

- Extend trickshot mode
- Check 9 Ball for missing rules
- Check 9 Ball rules for errors
- Properly handle balls that are shot off of the table
- Split gamestate into separate classes for 8/9-Ball and the Trickshot mode
- Add recommendation lines for good shots

All our ideas can be found in the TODO.md file in the game part of the project-documentation.

Simulator

Goals

- Realistic physical behavior of the balls
- Intuitive control
- Comprehensive code

Changes on physical properties

- Changed the bounce threshold of the simulator
- Changed the limiter bounciness and friction
- Changed friction of play area
- Added the weight, material, bounciness and friction to the balls
- Adjusted the shot strength

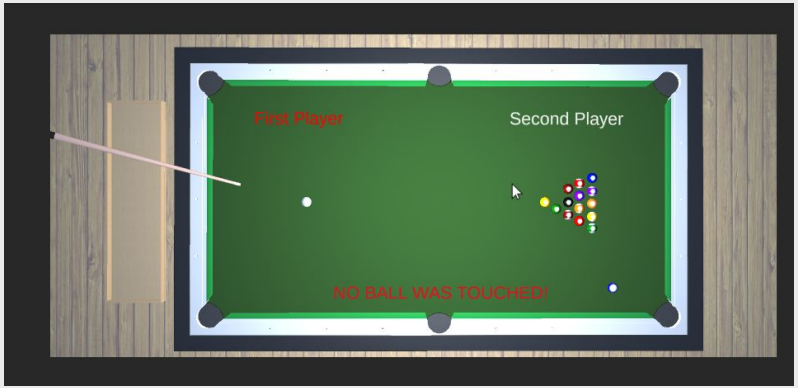
New Features

- Added free ball movement on RMB hold
- Locked shot direction on LMB hold
- Avoid shooting of the whiteball when UI buttons are clicked
- Balls respawn in Box, when they touch the ground
- Cue moves back on LMB hold and moves forward on releasing.
- Avoid charging of cue as long as balls are moving.
- Enabled the mouse input outside the billiard table.

Some Visuals

- Cue moves back when charging

- Shot direction is locked after LMB hold



Ideas for successors

- Shot strength could be regulated by mouse-distance to white-ball, not by LMB-clicktime
- Move cue backwards according to shot strength(currently the back movement is just linked to time not to shot-strength)
- Display shot-direction and first bounce-off direction

Tracking

Goals

Write tracking API to get all needed data for correctly displaying the billiard table

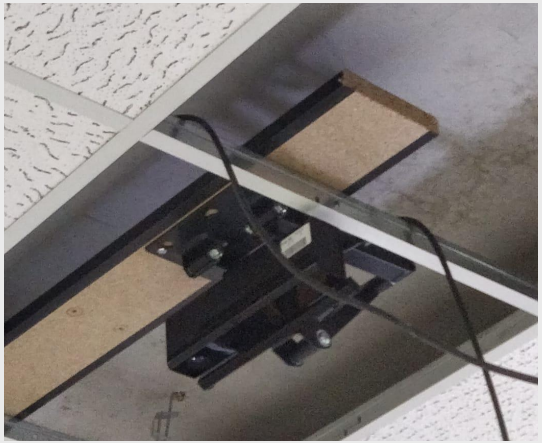
- Implement color tracking
- Implement depth tracking
- Implement collision tracking
- Use kinect data for tracking
- Simulate kinect input (testing)

Kinect

To use the Kinect data we firstly needed to add some advancements to the mount & table.

This included:

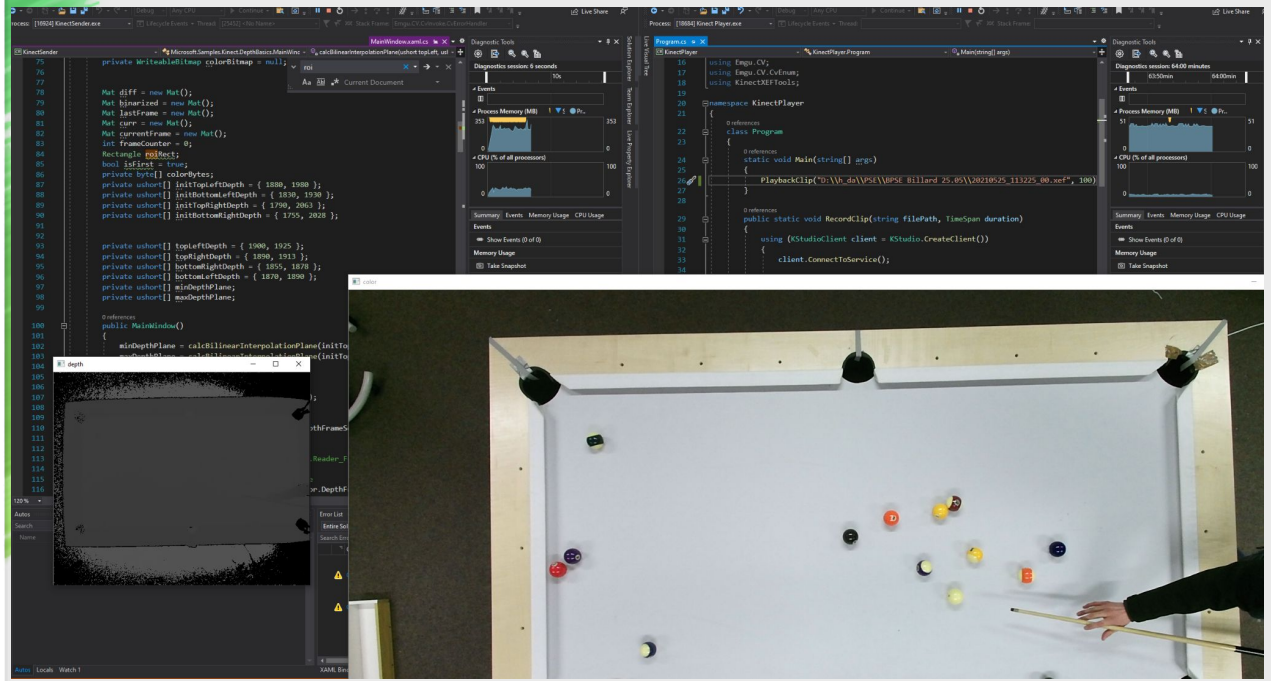
- Make a new mount to ensure evenness
- Level the table, also for evenness
- Position the table correctly so it is fully captured by the camera



Kinect

- After we had everything set up we could finally track some usable data.
- For this data to be used in our algorithm we needed to simulate the camera stream since we could not do this in the lab.
- We recorded an output stream containing depth and color data from the Kinect and used it for further development.
- In order to simulate the Kinect input we created our own solution in VS (Kinect Sender and Player)
- By playing this output you can simulate the color stream and depth stream as if the Kinect was connected via USB.

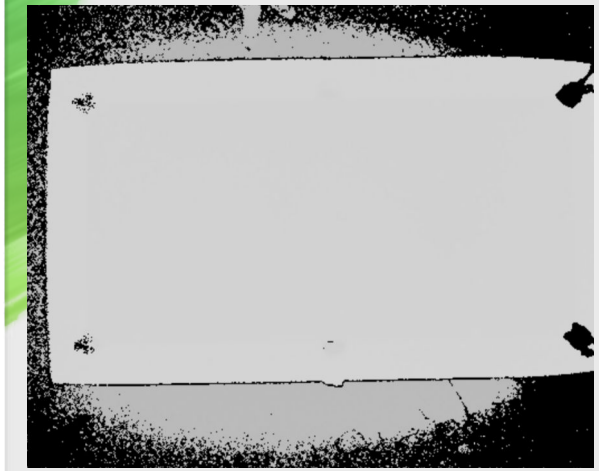
Kinect Simulation

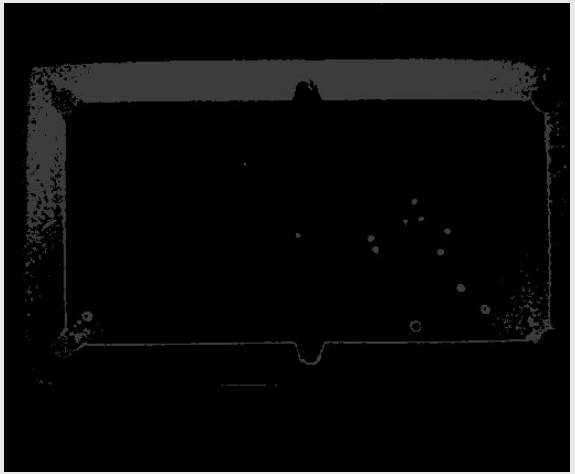


Kinect Depth

Step 1

- Use binary interpolation to remove noise from Kinect stream and set fitting threshold

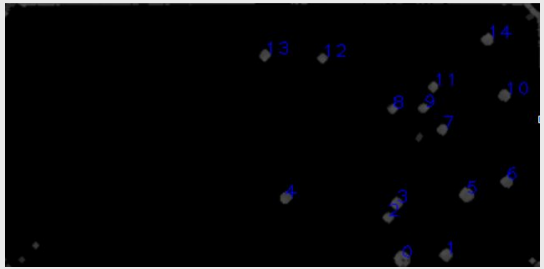




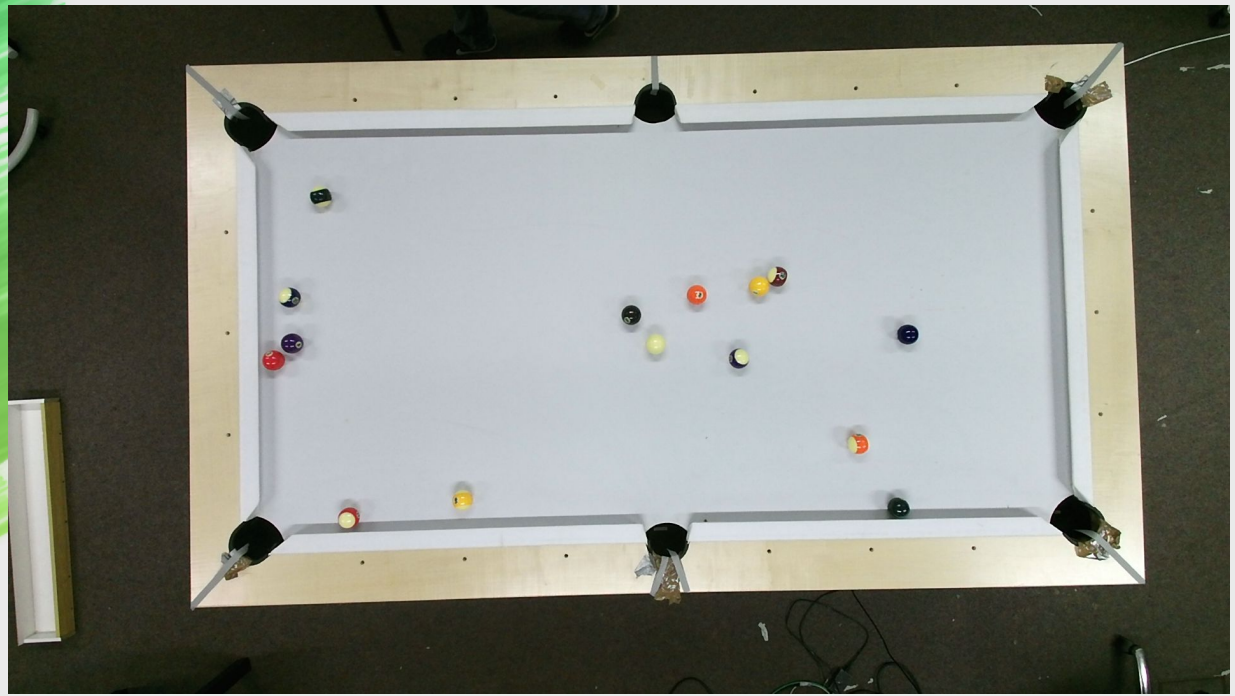
Kinect Depth

Step 2

- Find region of interest (upper left pocket to lower right pocket)
- Binarize the image and add contours with corresponding IDs



Kinect Color

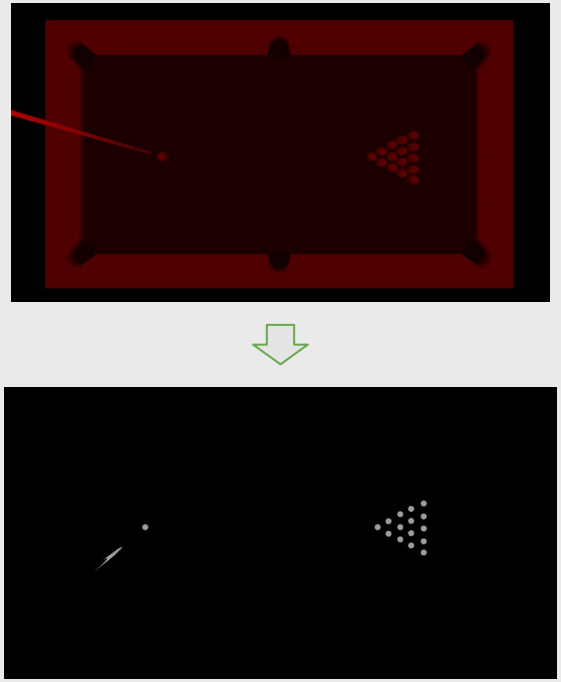


Unity Tracking

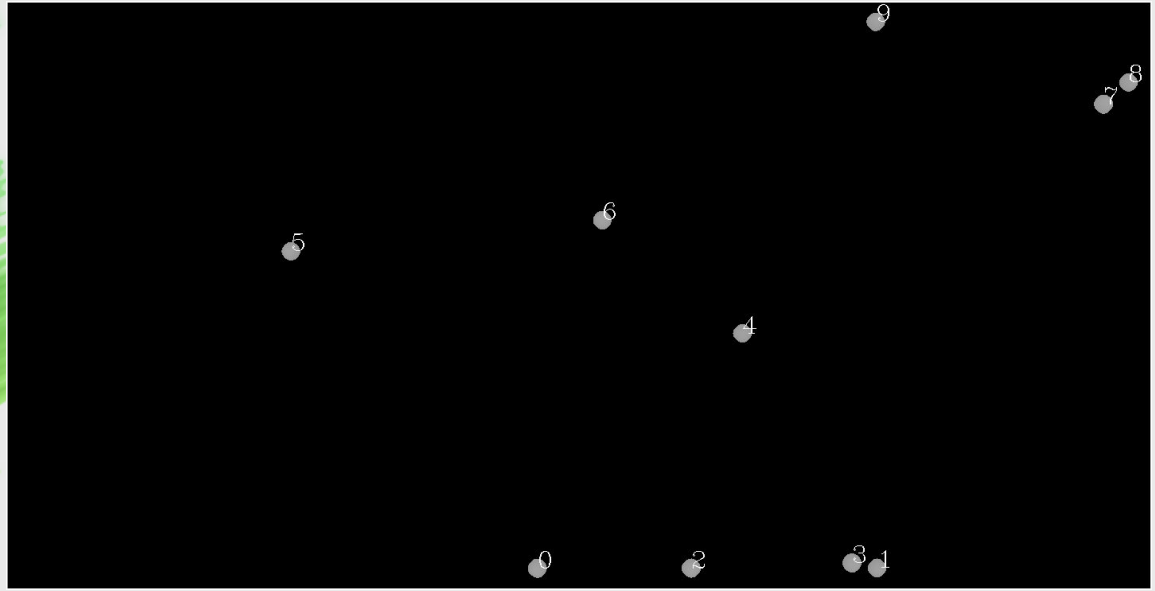
Since we want to use the camera tracking for both kinect and unity, the data we receive from unity needed to be the same as the kinect data

Unity Depth

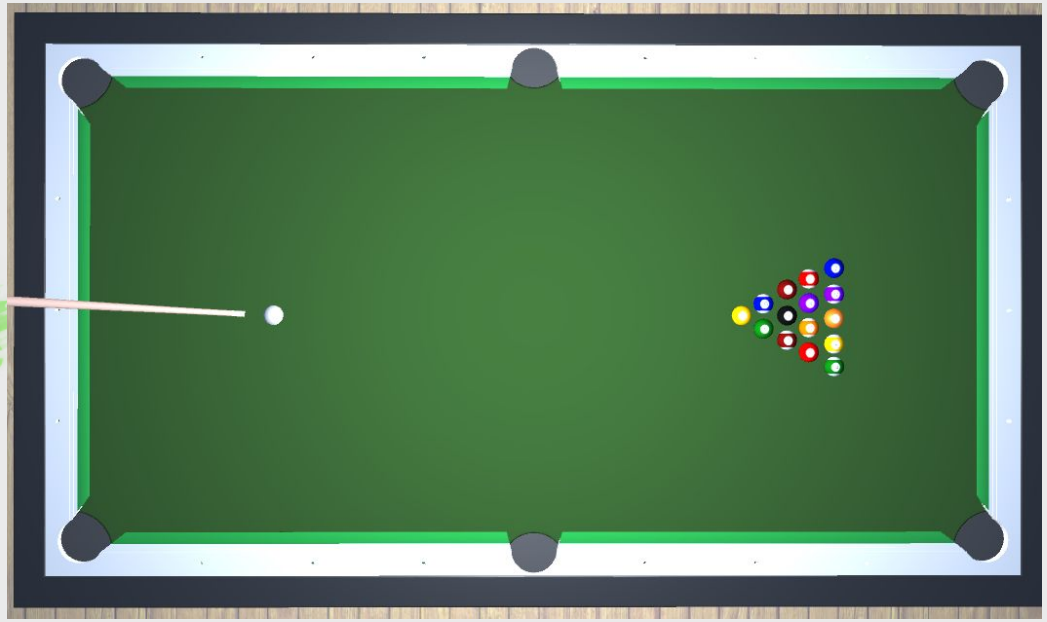
- Apply region of interest and binarization (no need for interpolation since table is perfectly flat)



Unity Depth



Unity Color



Unity Color

- Use positions of the balls acquired from the depth tracking to cut regions that include only one ball
- Evaluate the average color of these regions to determine the ball type using HSV

Collisions

- The Unity internal tracking now supports tracking of wall collisions.
- These are now used to fill the frame which can be used to build the scene.

Ideas for successors

- Integrate Tracking API into Unity
- Use camera tracking to track collisions
- Improve color detection for kinect data
- Add a functionality so that the camera tracking can use the kinect stream
- Improve performance

Effects

Goals

- **Main Goal:**
  - Get structure in the UI-system and improve the user experience
- **GUI:**
  - Design and implement new elements/features
  - Support other Teams to create and implement UI elements with functionalities
- **Sound features:**
  - Record and modify sound effects
  - Insert and adapt sounds to the Project

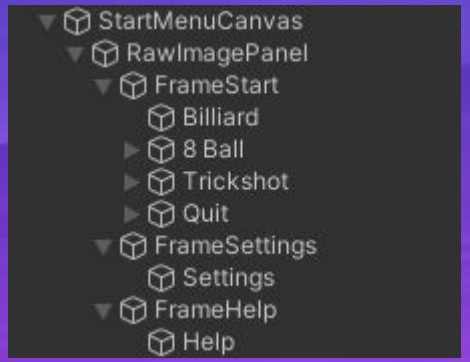
GUI

- Structure UI components to simplify the adding of new elements
- Adjust text colours
- New UI menu for Trickshot Mode
- Unify existing menus for a uniform graphical visualization

Visualization

- Structure UI elements (example Start Menu)
  - Canvas with background image
  - Frames with layouts to structure elements

➔ New elements are kept within the Frame and all sizes adapt to new elements



Visualization

- New Menu for Trickshot mode
  - all needed inputs for the game logic

Visualization

- Pockets are connected to the dropdown menu

New sound effects

- New Background Music
- Cue-Shot Sound
- Player-Switch Sound
- Ball-Hits-Wall-Collision Sound
- Sound for a ball that falls into a pocket

Known bugs

- While the sound of the cue that hits the white ball is played, the sound when a ball hits the wall will not be played perfectly

Ideas for successors

- Find a method that the Cueshot-Sound and then the ball hits wall sound are played every time perfectly
- Add Sound and animation for winning
- Add a Presentation of a leaderboard
- Visual representation for the remaining balls for each team
- **Further into the future:** Install an LED tape around the table where segments can be controlled to visualize a ball rolling at the wall with light

Display

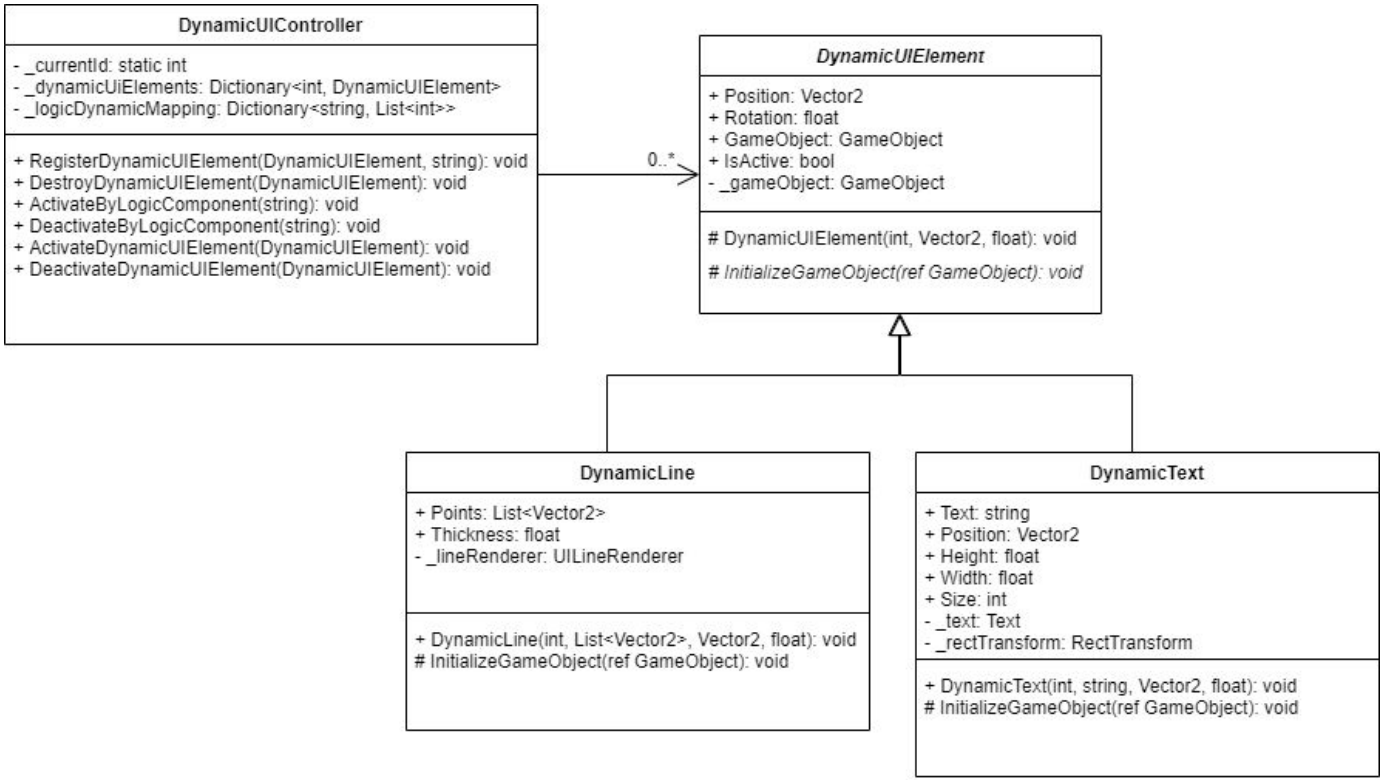
Goals

Make an easy to use component to display different types of UI elements on the billiard table.

- Support for…
  - Dynamic UI created and altered at runtime (e.g. lines or text)
  - User-Interactable static UI created before runtime (e.g. menus)
- One code base for the simulator billiard table and the real world billiard table
- Separate scene with only the display → no simulator

How it works

Dynamic UI


DynamicUIController
- _currentId: static int
- _dynamicUiElements: Dictionary<int, DynamicUIElement>
- _logicDynamicMapping: Dictionary<string, List<int>>
+ RegisterDynamicUIElement(DynamicUIElement, string): void
+ DestroyDynamicUIElement(DynamicUIElement): void
+ ActivateByLogicComponent(string): void
+ DeactivateByLogicComponent(string): void
+ ActivateDynamicUIElement(DynamicUIElement): void
+ DeactivateDynamicUIElement(DynamicUIElement): void
0..*
DynamicUIElement
+ Position: Vector2
+ Rotation: float
+ GameObject: GameObject
+ IsActive: bool
- _gameObject: GameObject
DynamicUIElement(int, Vector2, float): void
InitializeGameObject(ref GameObject): void
DynamicLine
+ Points: List<Vector2>
+ Thickness: float
- _lineRenderer: UILineRenderer
+ DynamicLine(int, List<Vector2>, Vector2, float): void
# InitializeGameObject(ref GameObject): void
DynamicText
+ Text: string
+ Position: Vector2
+ Height: float
+ Width: float
+ Size: int
- _text: Text
- _rectTransform: RectTransform
+ DynamicText(int, string, Vector2, float): void
# InitializeGameObject(ref GameObject): void

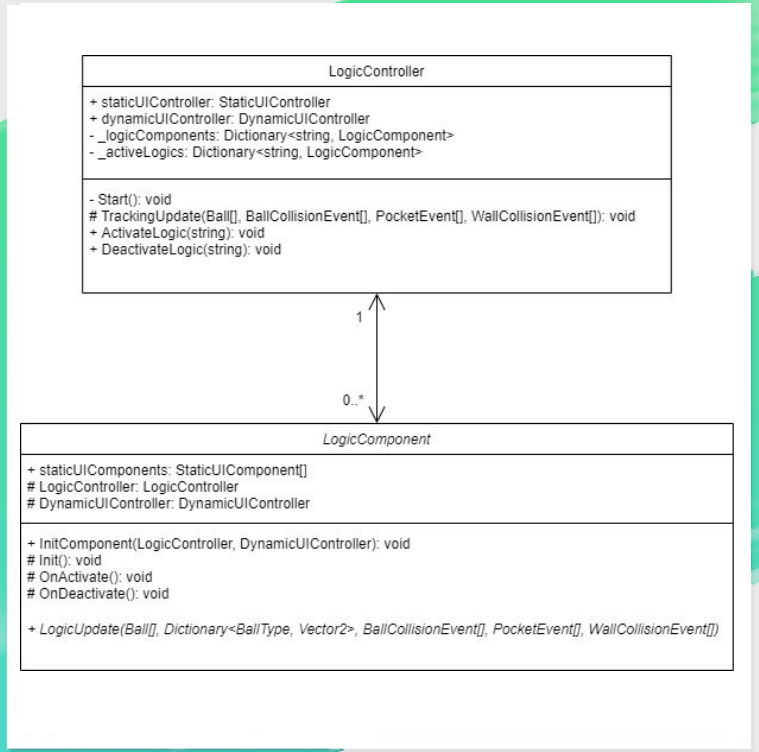
Logic

Logic Controller

- Receives tracking data
- Distributes tracking data to active logic components

Logic Component

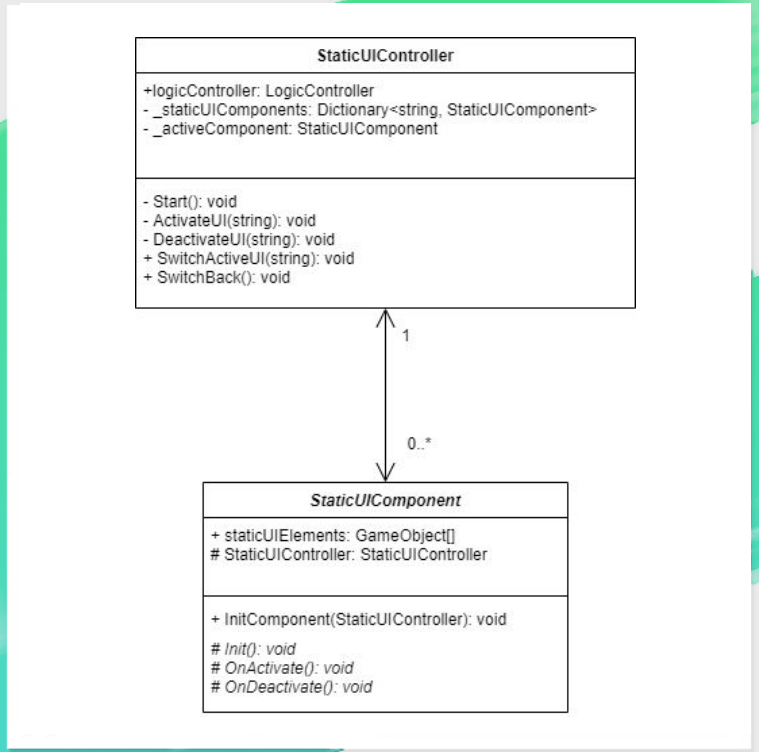
- Decides when and where to display dynamic UI elements based on tracking data



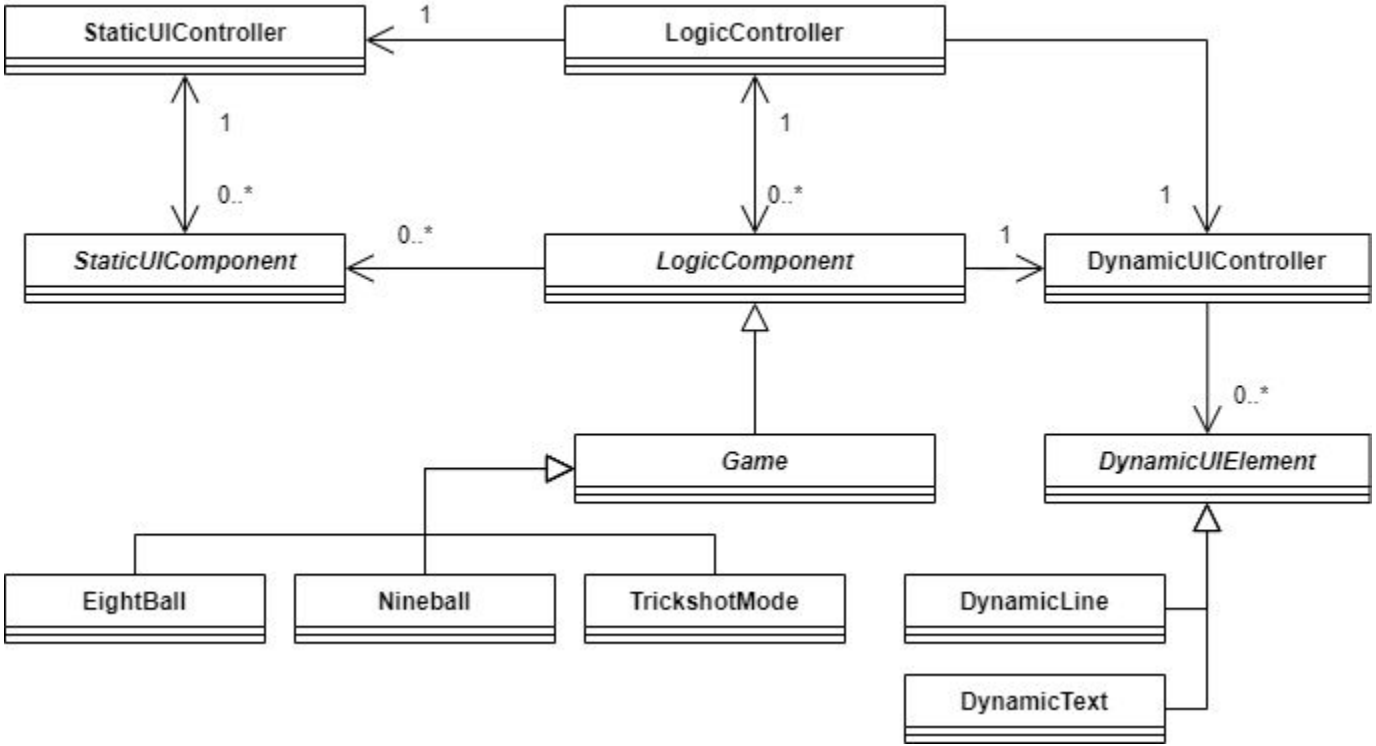
Static UI

Static UI Controller

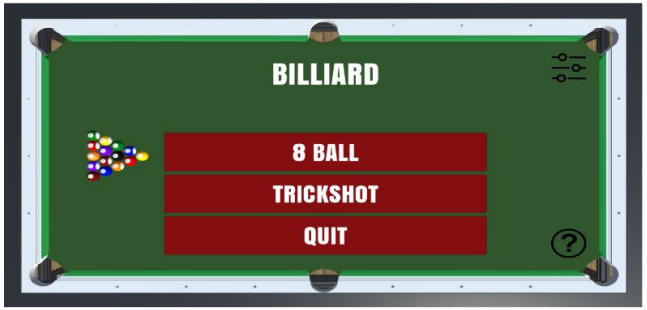
- Controls what static UI component is currently active
- Memorizes order of UI switches



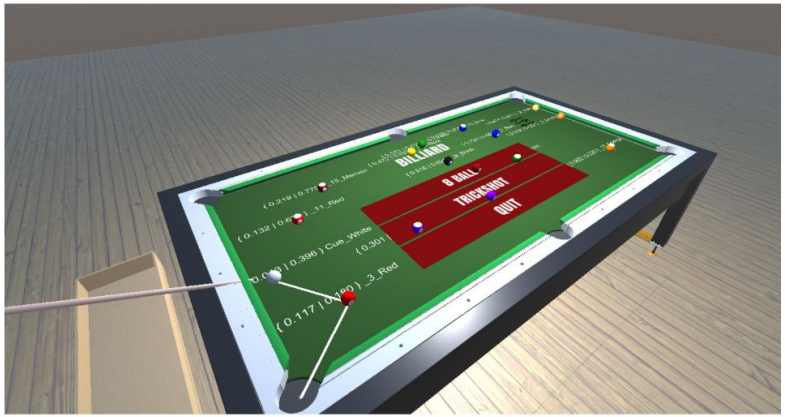
How its all connected



Before



After



Ideas for successors

- Refactoring the code base
- Add kinect receiver to display scene
- Adapt size of DynamicUIElements to screen size
- Add proper support for sound in the display prefab
- Add layer support for dynamic UI

PROJECT SHOWCASE

Our problems and solutions

Problems	Solutions
Tasks or responsibilities are vague	Be more precise, clear up uncertainties
People do not follow the rules	Precisely describe the rules and make people read them
People cannot solve their Git problems	Experts provide help
Time-management failed	Reserve buffers; try to finish earlier than the deadline
People are not communicating enough with each other	Hold frequent meetings outside of Sprint Review/Retrospective
People do not actively participate in meetings	-

Recommendations for our successors

- Make sure **everyone knows their tools**
  - Git: basic usage and problem-solving
  - Unity: understand the editor and engine
- **Be precise** when communicating tasks and responsibilities
  - WHAT has to be done HOW?
  - WHO should do it?

Conclusion

- Finished our Main Goals

- Main Focus Points
  - Communication
  - Time Management
  - Simple Setup for easy introduction into the Project
  - Asking for help is good, but trying yourself before asking goes a long way understanding the issue and solution