

Brandon Liu, Eric Chien
CPSC 490 Senior Project
May 11, 2022
Final Report

Gammix: Mixed-Reality Multiplayer Board Games

Motivation

Many of us have played games like chess online. The online versions of multiplayer board games are fun, but lack the tactile experience of physically moving game pieces and seeing the board and gameplay live in 3-D space. But what if you could still play with others online, but use the real-life, physical board game in a straightforward way? To explore this question, for our senior project we designed and built a system called Gammix for such mixed-reality multiplayer board game experiences.

Acknowledgments

We would like to acknowledge the professors and advisors who have helped us get to this point today. Thank DUS Richard Yang, senior class advisor Prof. Drago Radev, and CPSC 490 coordinator Inyoung Shin for organizing and making the senior project possible. We'd also like to acknowledge Prof. Marynel Vazquez, Man-Ki Yoon, Prof. Aaron Dollar, and Andrew Morgan for their advice early on in our senior project. We would also like to express our gratitude to our early professors in computer systems, Prof. Abhishek Bhattacharjee, Prof. Zhong Shao, Prof. Rajit Manohar, and the late Prof. Stan Eisenstat, for providing us with the fundamentals of computer science and ample opportunity over challenging semesters for the two of us to work together and become friends. In spring 2020, after everyone went home due to the pandemic, the two of us stayed in contact, attending Prof. Abhishek's CPSC 323 lectures and bouncing ideas off of each other for the psets. There was a distinct moment, perhaps over Zoom, when the both of us simultaneously realized that it was seeing each other continue to

work hard in 323 that pushed each of us individually to continue challenging ourselves (and not just take the easy route, given that it was all pass/fail during remote classes). In that way, we found a recursive loop of motivation, and we certainly have each other to thank!

Lastly, we would like to say our sincere thanks to our advisor Prof. Lin Zhong for being an especially thoughtful, patient, and supportive professor and mentor. Brandon would like to thank Prof. Zhong in particular for the opportunity to work with him in the Efficient Computer Lab in the summer of 2020, after his internship was canceled due to the pandemic. That summer proved an especially formative experience in his computer science education. We appreciate Prof. Zhong's time, advice, and ideas — like combining computer vision with a robotic arm for our project — which we ended up very happy about as the process and results were exciting to us and we both learned a great deal.

Introduction and demonstration

Gammix brings the offline board games experience online by using a robotic arm to recreate opponents' actions on your physical board. Each move is captured using a standard webcam, interpreted using computer vision, then sent over the internet and replicated using a robotic arm on the opponent's boards. This system differs from our original proposal. Originally, we proposed using a projector to display an interface showing the opponents' moves atop the board, thus allowing the player to easily update the board state based on it. Instead, we have designed and built our final project around a robotic arm as the actuator, such that the opponents' moves are carried out automatically. This way, gameplay is not interrupted by the player having to update the board based on their opponents' moves, as it is done automatically, so the overall experience is much more seamless.

There were multiple key challenges and open questions at our project's outset: 1) whether a single standard webcam could accurately detect the board game state, and what computer vision algorithms would help 2) how best to represent the board state for multiplayer

game purposes, 3) how to design a software architecture to synchronize new actions with potentially multiple remote opponents, and 4) how to translate pixel locations from the webcam frame to the robot arm frame, so that the arm knew where to move a piece and could actually do so when it received commands. We were able to solve all of these challenges by the end of the semester and will detail how we came to our specific solutions in this report.

We implemented Gammix and demonstrate its feasibility for the board game Settlers of Catan, using a standard USB webcam and Trossen PincherX-100 robotic arm. The system works end-to-end: after the first player makes their moves on the board and ends their turn, Gammix detects their moves locally, then issues the commands over the internet for the opponents' robot arms to replicate them remotely. Linked in the below screenshot is a short video to demonstrate this process.¹



In the first few seconds, you can also see an image of all vertices detected and represented atop the image of the board. This was a key goal of our computer vision pipeline, or move detection module, which Brandon designed and implemented. Eric was responsible for the programming and providing an interface for the robotic arm to accept moves. Both of us collaborated throughout and iterated frequently on our software design, APIs, and choice of data

¹ <https://www.youtube.com/watch?v=z9rEOX2ND-E>

representation, to connect our two halves of the system. Our system works for an arbitrary number of turns, as well as an arbitrary number of pieces placed per turn, and currently supports placing settlements. Placing roads, upgrading to cities, and moving the robber are left for further work, due to time constraints in one semester. However, the key challenges of detecting, representing, and actuating moves have been addressed, making implementing these additional pieces relatively straightforward after understanding our design.

Prior work

Catan-omous Dealer is an autonomous dealer system for distributing resource cards in the board game Settlers of Catan, built as a capstone project by Carnegie Mellon students.² It uses computer vision to detect and track the board state and an array of motorized card dealers to distribute resources. While Catan-omous Dealer is effective at detecting which players are adjacent to what resources, it doesn't identify the precise location of pieces in the webcam frame and translate that to the world frame, which is what Eric and I implemented together in order to solve the problem of replicating a detected move on the Catan board. We use the same first two algorithms (Canny edge detection and watershed) in our own computer vision pipeline, but take the project in a different direction by identifying vertices so that we could get coordinates. Moreover, Catan-omous dealer relied on modifying the game pieces and a specialized platform to set the board upon, while our system required no such modifications or custom hardware.

Another prior system that relates to our project is Catan Image Recognition, which also detected Catan's board state using computer vision with varying reliability but without any further game actions. As it was open source, we were able to reference a portion of their code for the isolation of the board from the background in our project.³

² <https://course.ece.cmu.edu/~ece549/spring16/team08/website/pages/design.html>

³ <https://github.com/Vieja/Catan-Image-Recognition>

Prior projects have programmed robotic arms to play Catan locally, as shown in a Kuka Robotics' video with students in OTH Regensburg in Germany.⁴ But, base on our research, no other prior work has used a robotic arm to play Catan with remote opponents, over the internet.

We were also inspired by the system built by engineer Shane Wighton that allowed him to play pool against other players online but on an actual pool table.⁵ It did so by projecting an interface onto the pool table, and by employing a custom-built robotic pool cue that automatically lines up and makes the shot with the force and angle specified by the online player. While Wighton initially built the robotic pool cue and projector system as a way to guarantee that he sinks every shot algorithmically using software, the added benefit of his system enabling an online multiplayer experience on the physical table turned out to be the real highlight and a source of joy.

High-level software design

Our software design consists of two main modules that run on each players' local machine: one for move detection and one for move actuation, with message passing via a singular new_moves topic with a pub/sub model. The move detection module checks if it is the current player's turn, and if so it waits for the player to finish making their moves. Once the player has completed their turn, they press space, and the computer vision node will process the webcam images to detect the type and position of any new moves they made. This information is published to a topic called new_moves, to which all move detection and move actuation modules subscribe to. Upon notification of new moves, the move actuation module on each machine checks if the moves are from an opponent, and if so, plans and executes each move by sending commands to its corresponding robotic arm. When the move detection module receives new moves, it ignores the move data itself and just updates its local turn counter.

⁴ https://www.youtube.com/watch?v=oCQPWv_ky2c

⁵ <https://www.youtube.com/watch?v=vsTTXYxydOE>

We use the pub/sub architectural design pattern as it provides a straightforward way for multiple subscribers to be made aware of new moves and thus changes needed to the board state. Our design provides a minimal interface for turn-based multiplayer games to be implemented. In figure 1, we provide an overview of the Gammix architecture. We show the Gammix design's viability by implementing it for the popular board game Settlers of Catan. In this implementation, figure 1's rectangles correspond to Python programs/ROS nodes, diamonds correspond to hardware, ovals correspond to topics, and arrows represent sequential actions/communication. Blue arrows represent publishing, and red subscribing. In figure 2, we show an example of how this architecture progresses over a turn for each player in a three-player game of Settlers of Catan.

Figure 1. Gammix Architecture Diagram

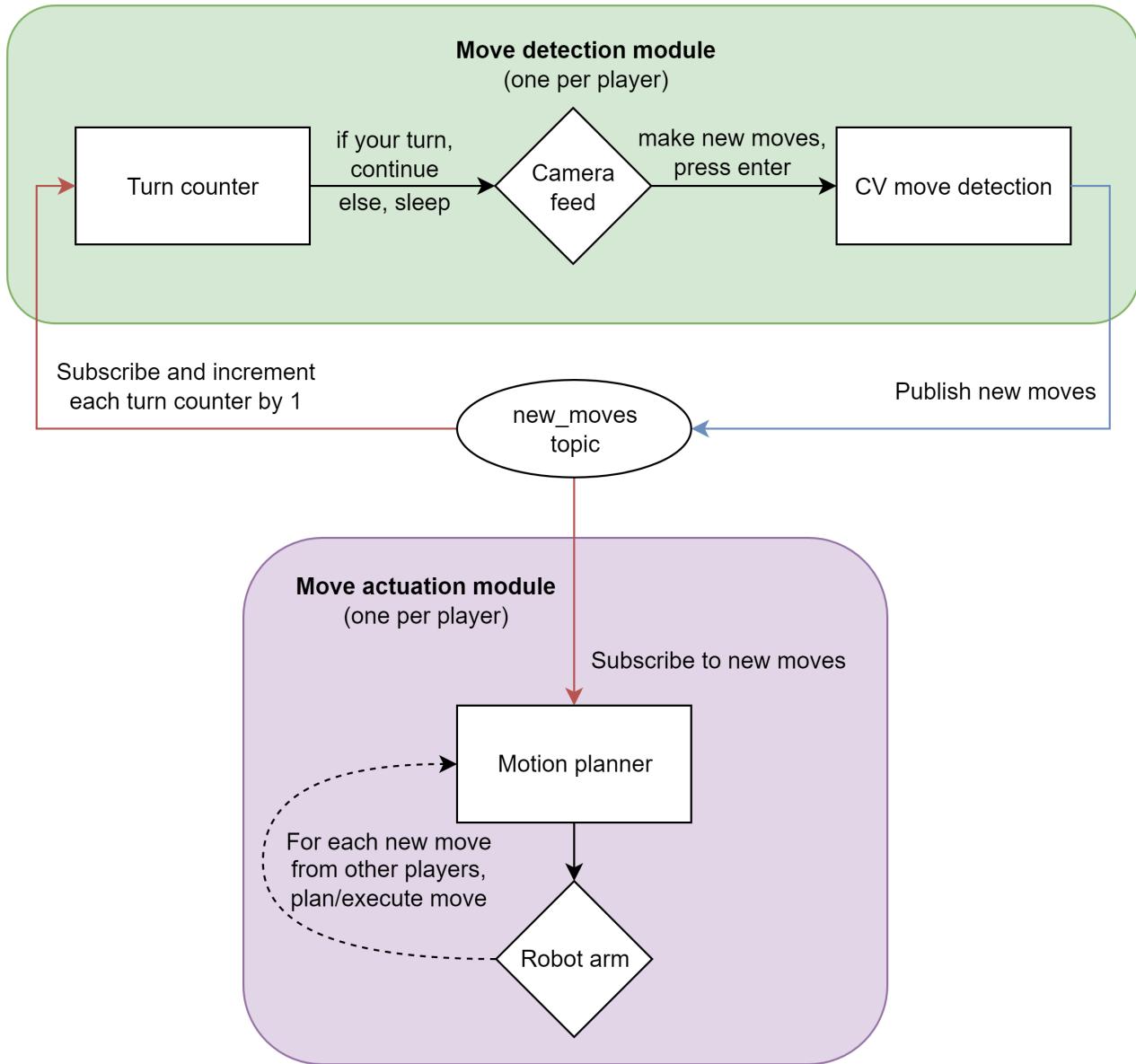
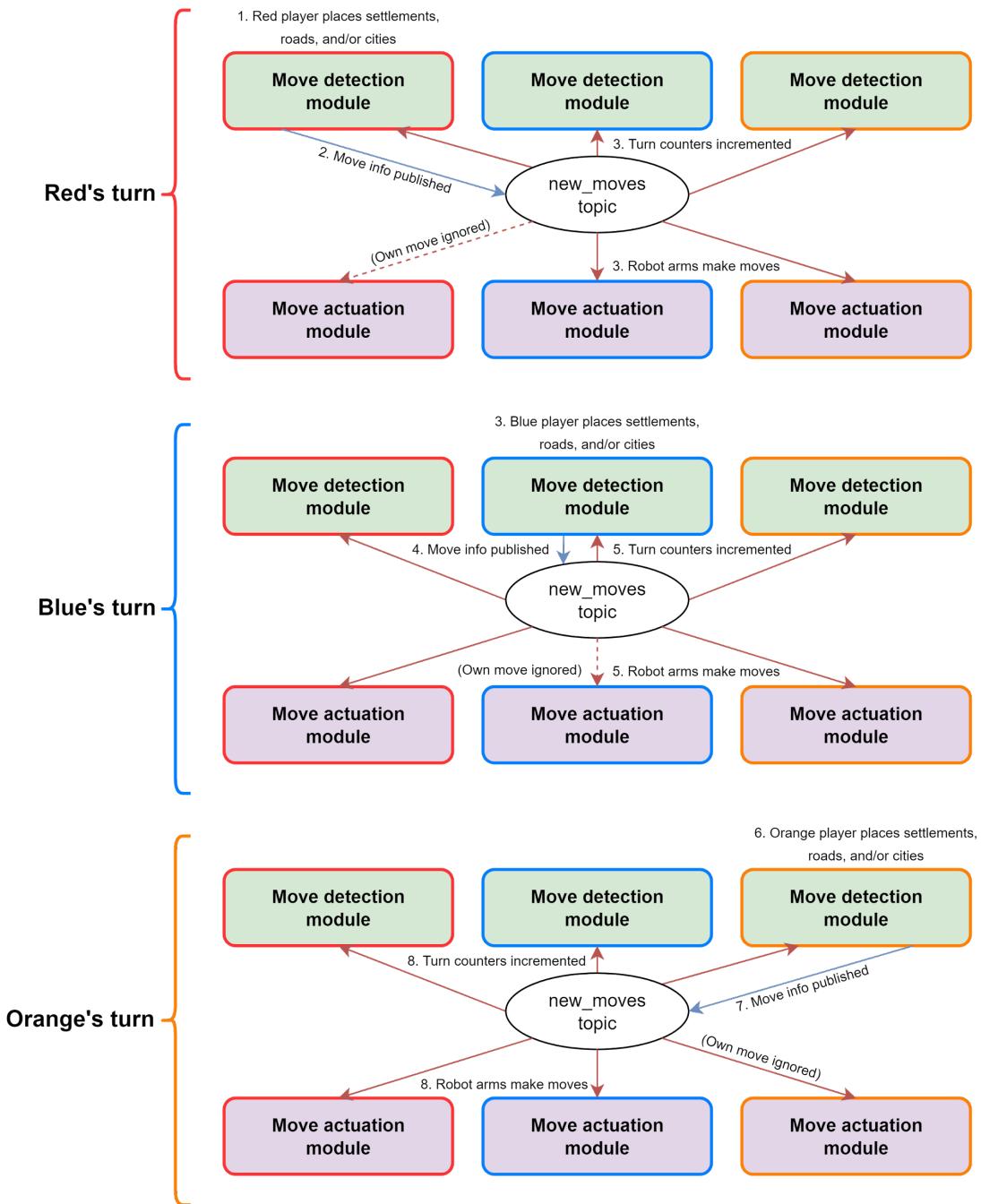


Figure 2. Three turns in a three player game of Gammix Catan



Board and piece representation/design

One of the central questions our project wanted to solve was how we could translate information of the pieces on a board from a camera to the information needed for a robot arm to

make those same moves on a different board. As both the reference frame of the camera and the robot arm are drastically different — the camera frame holds a pixelated bird's eye view of the board, while the robot arm must use relative (x,y,z) offsets from the base in order to move objects — a clear system of synchronization between the two was needed. To accomplish this, we first needed to construct the representation of a board in both the frame of the camera as well as the robot arm. By having the concept of a board in each reference frame, we are able to create a framework to then understand what it means for a piece to be placed on the board. Thus, for each frame (besides the linking pub/sub frame) we have a preprocessing step to generate the board representation and an execution step that handles the translation of piece placement from camera to robot arm.

We began with the camera frame. Since we knew that pieces could only be placed on vertices and edges, we only needed to maintain a list of a finite number of points in an image. Our understanding of what it means for a piece to be placed would then simply be the idea of checking to see which vertex or edge point coincides with the point of a piece on the image. Thus, for the camera frame, we represent the board as a list of pixels containing all possible positions that which a piece could be placed. Similarly, with this board representation, to represent the placement of a piece, we simply detect which pixel the piece was placed on. The next important frame we will discuss is the robot frame. The robot arm needs to understand a similar list of vertices/edges as the camera, but rather than using pixel coordinates, the robot arm needs to understand a list of physical offsets representing each valid placement of a piece. By having a list of physical offsets we are able to utilize inverse kinematics to plan the movement of the arm to each position of the board. Thus, the board is represented as a list of offsets. Using this board representation, we represent the placement of a piece as a selection of the exact offsets needed to place a piece in the specified location.

Now that we have established the concept of a board for both the camera and robot arm, we now need a concrete way to translate from one board representation to another. So, we

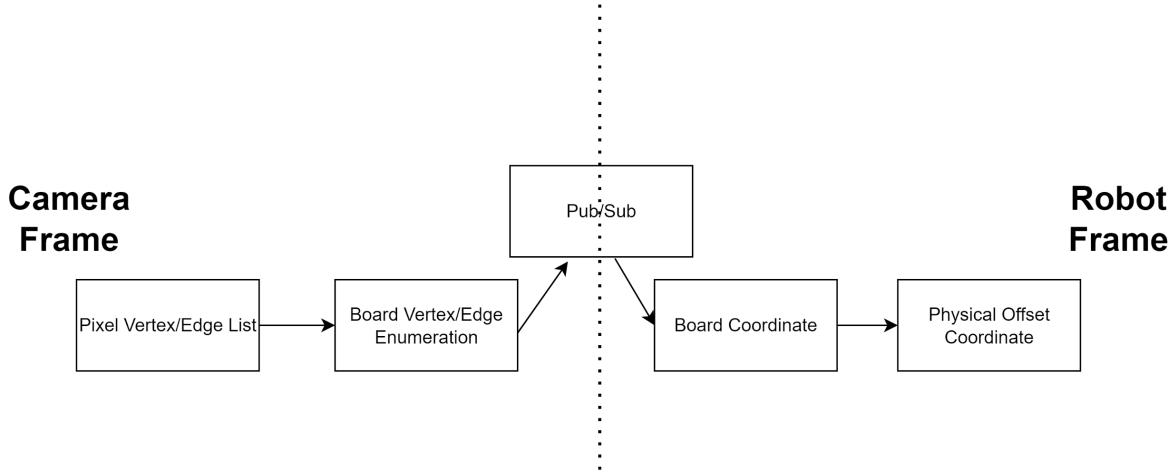
need a series of transitory frames. Let's start with a camera transitory frame. While our vertex/edge pixel list from the camera frame now provides us the power to represent the important features of the board, we still have no idea how each vertex/edge relates to each other. As in, given a pixel vertex/edge, we have no idea if this represents a vertex/edge at the top of the board, leftmost bottom corner of the board, middle of the board, or any other positional information that the robot arm placing the piece would need to understand. Thus, we need to create a board representation consisting of an enumeration of vertices/edges on the board with a translation dictionary converting a pixel to a specified point in the robot arm frame. To represent the placement of a piece, we find the nearest valid board position from the given pixel where the piece was detected.

Similar to the camera, we also need a robot transitory frame. This frame represents the board as another sort of enumeration of points on the board, so that we can map board positions to physical offsets, and we can have a notion of the ordering of the physical offsets on the board. Note that even though both transitory frames utilize board positions, the representation of the board positions can and in our case does vary between the two frames. This is done for convenience purposes which will be explained later in the report. But what this does mean is that the robot transitory frame has an extra role of translating from camera board position to robot board position. The notion of placing a piece is thus the translation of camera board position to robot board position and then subsequently translating robot position to robot offset to represent where the piece would be placed relative to the robot arm. Finally, to connect the two systems together, we have a pub/sub frame that simply acts as a messenger from camera transitory frame to robot transitory frame.

Therefore, we have described the major components of our 5 stage board and piece representation pipeline. Let's recap and put it all together. Before any piece processing begins, both the camera and robot frames must do preprocessing to generate their board representation lists. The camera transitory frame must also generate the pixel to board enumeration dictionary

while the robot transitory frame must generate dictionaries for both the camera board enumeration to robot board enumeration and robot board enumeration to physical offset. After the preprocessing is finished, we can demonstrate the process of translating the position of a piece from placement to robot arm movement. To begin, a piece is placed on the board. The camera frame then calculates the pixel of the piece and determines the vertex/edge that the piece is covering by comparing against the vertex/edge pixel list. The camera transitory frame then takes the pixel and converts it to an vertex/edge enumeration which is sent by pub/sub to the robot transitory frame. This frame converts the enumeration to a robot board coordinate and subsequently converts the robot board coordinate to a physical offset. Finally, the robot utilizes the physical offset to move a piece to the same position as captured by the camera.

Figure 3. Board and Piece Representation Pipeline



Board Representation Implementation

For reference, the code for our implementation is hosted at:

https://github.com/RollingPeanuts/senior_project

- src directory contains the source code python scripts for our implementation
- utility directory contains helpful debugging scripts

- `start_arm.sh` is a script for running the necessary startup bash commands to start the robot arm

Camera Frame

TODO: Talk about preprocessing

- Write about the implementation of the camera module and the decisions made to translate image to vertex list.
- Describe the process of using a dfs to create a list of edges from our list of vertices.
- Maybe include the below (after the next todo) paragraph segment if you think it'd be interesting to talk about

TODO: Talk about the execution of the program

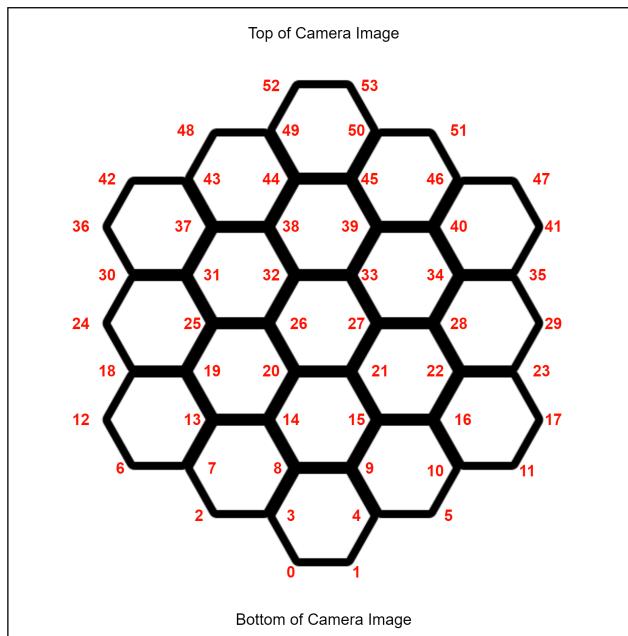
- How the diff detection works, what is the process of going from placing a piece to having it detected and getting us the pixel coordinate (This is basically just taking a picture and subtracting the two pictures to find the difference. Then thresholding to remove noise and finding the bounding rect to determine the center).

While it may be tempting to assume that we could simplify the algorithm by always detecting two neighboring fixed points on the board (like detecting the pixel coordinate of the lowest vertex of the board) to determine the translation from pixel distance to physical distance to extrapolate all other pixel coordinates of the board, we need to account for the potential camera angle skew that is very likely to exist in the image. Thus, while the hexagons on the board may be regular, the image may not represent the hexagons as regular, so hexagons closer to the camera may have a larger side length than hexagons further away, creating potential error and limiting the ability of the application to generalize the board representation problem.

Camera Transitory Frame / Board Enumeration

At this point we are given an unordered list of pixels corresponding to all of the various edges and vertices filtered out from the image. With this list, we are able to spatially understand where each point is located relative to each other, however, we need a little bit of outside information in order to understand how the points overlay onto the board. That is, we need at least one fixed point with known relative positioning of both the board and pixel list. To accomplish this, we orient the board such that the top and bottom edges of each hexagon in the hexagon grid form horizontal parallel lines relative to the image captured by the camera.

Figure 4. Board orientation relative to the camera with vertex enumeration



This provides us a natural enumeration of vertices and edges by ordering them bottom to top with left to right handling tie breakers as shown in figure 4. Now naturally, we cannot always expect that our orientation of the board would always be perfectly parallel to the camera orientation, so it's possible for slight deviations to occur like the 0th vertex being slightly higher than the 1st vertex. Therefore, we can't simply sort by y coordinate and tie break on x coordinate. Instead, to have a degree of fault tolerance, we devise an algorithm for numbering

by enumerating one row at a time. Notice that the first two rows have 2 vertices and 4 vertices respectively, the last two rows have 4 vertices and 2 vertices respectively and all other rows have 6 vertices. We first sort our list of vertices by lowest y coordinate. Then for each row, we take the first n vertices of the list (where n is the number of vertices in the row), and go from lowest to highest x coordinate to give each vertex an enumeration. We then remove all vertices processed from the list. This ultimately deals with any minor tilting the board may have. It does create issues when the list of edges and vertices received from the camera is slightly incorrect (missing vertices/edges or falsely identifying points that aren't really vertex/edge) as we are simply taking the lowest 54 pixels in our list without any error correction. But, such correction code is currently outside the scope of our project and would be future work that could be implemented in a future iteration of the project.

We can employ a similar system for creating an enumeration for the edges. Now, we have two distinct dictionaries, one containing pixel to vertex number translation and another containing pixel to edge number translation. We can effectively combine both dictionaries into one, making sure to add an additional field to each value stating whether the pixel to point translation yields a vertex or edge. This completes the preprocessing step of this frame.

Later in the execution of the program, when this frame is provided with the pixel coordinates of a piece being placed, we simply need to find the nearest point to the piece pixel.

We can simply use the distance function $d = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$. We find the distance of the piece pixel to the pixel coordinate of each point and can run a linear time comparison to find the closest point. With the closest point determined, we can send the vertex/edge enumeration through pub/sub to the robot arm to process.

One potential optimization that could be addressed as future work would be to implement a k-d tree for the nearest neighbor search. While the space complexity would be $O(n\log n)$, we could reduce our search to $\log(n)$ (as opposed to $O(n)$), a potentially reasonable

improvement as there could theoretically be an infinite number of piece placements. That being said, realistically the operation of the application would most likely be bottlenecked by the physical movement of the robot arm much more than the calculation of the distance of the piece to other points.

Pub/Sub

The pub/sub component of the pipeline can be simply imagined as a message delivery system across the internet between a camera handler to the robot arm handler of each other player. As we utilized the google cloud pub/sub api, the code on our end was rather simplistic and just consisted of ensuring that the proper authentication keys and topic names were used as well as any of the other setup steps required. The implementation for the publisher in the camera module was thus just converting the point enumeration to a string and then to a byte to be published to the cloud. For the subscribers in the robot arm module, we utilized a callback function to ack the message and read the message to unpack piece location points. Finally, our message to be sent over follows the format: "<piece_type> <board_enum>, <piece_type2> <board_enum2>" where piece_type is a string representing the type of piece to pick up and board_enum is an integer representing the position to place the piece. Each action is a collection of two values separated by a space, piece_type and board_enum, and each action is delimited by a ','. Thus, our program supports an indefinite number of actions being sent in one command. Currently, piece_type can be any string as its nonfunctional and we don't currently have the infrastructure set up to differentiate between piece types. This is another feature that would be interesting to implement in the future.

Robot Transitory Frame / Board Coordinates

The central idea behind our robot board coordinate system is that it simply takes a single calibrated point with known offset and known position on the board to determine the required

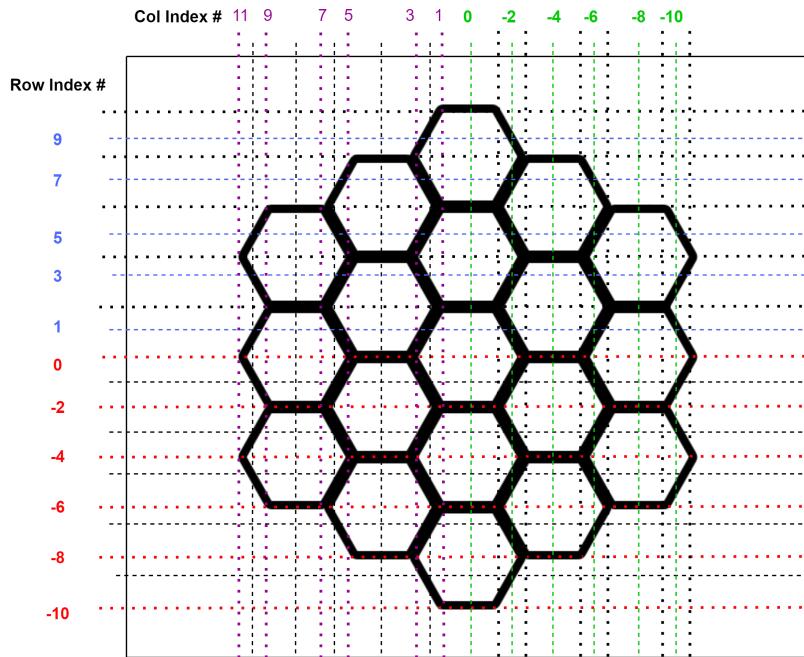
offset of all other points on the board. Unlike the camera frame, there is no notion of perspective skew muddling our sense of distance, rather we can just use the fact that we have a regular hexagon grid to calculate the distance between two arbitrary points on the board. Thus, in designing a robot board coordinate system, we wanted to create an intuitive representation of the board in which we would easily be able to determine the distance between any two points on the board. In creating the coordinate system, we leaned heavily onto the fact that we were working with regular hexagons. There was clear uniformity in the vertex and edge distribution of the hexagon grid, along both horizontal and vertical axes. For this reason, along with the fact that the offsets required by the robot arm required (x,y,z) offset values, we decided to pursue a cartesian coordinate system using a square like grid system.

The first decision we made that helped to constrain our design was to set the orientation of the board relative to the robot arm to be the exact same as the orientation of the board relative to the camera. That is, the same two vertices at the bottom of a camera image are the two closest points to the robot arm. By enforcing this orientation, we not only simplify the conversion process between camera board position and robot board position, but also create a uniform horizontal axis to work with. Now any consecutive two points in the same column (same x coordinate) are a uniform distance apart, let this be a distance x . If we draw horizontal lines for each vertex and edge point in the hexagon grid, we are left with lines that are each $x/2$ distance apart. This forms the basis of the row numbering system for our square grid overlay. Finally, since we know that there is a horizontal symmetry in the hexagon grid, we set the center horizontal line to be row 0. With this setup we now have two important properties, for any four points $a=(x, y)$, $b=(x, y+1)$, $c = (u, v)$ and $d = (u, v+1)$, $\text{dist}(a,b) = \text{dist}(c,d)$; also for any two points $a= (x, y)$, $b = (x, -y)$, $\text{dist}(a, (x, 0)) = \text{dist}(b, (x, 0))$. This effectively means that to get the y component difference between two points, we simply take $((y_2 - y_1) * \text{hexagon_constant})$.

This same framework can be applied to the x axis as well, although the conversion is not as perfectly nice. To begin, we once again take the vertical line crossing the center of the

horizontal grid and set it to 0 to enforce the symmetry. If we draw a vertical line for every vertex and edge, we see that every fifth line is uniform with each other, but the four lines in between are not uniform with each other. Thus, to determine the distance between two points, we calculate how many uniform 4 line widths are between the two points and add a custom mathematically calculated offset depending on the remaining non uniform distance. Figure 5 shows the complete board coordinate mapping in action. This then establishes a cohesive representation of the board that allows us to conveniently calculate the relative distance between two points. Thus, once we have a single calibrated point, we can calculate the specific offsets relative to the calibrated point of the point we want to put a piece at.

Figure 5. Board Coordinates Mapping

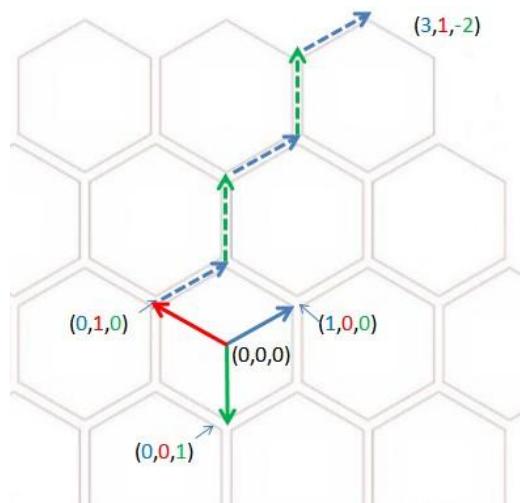


One potential alternative to the grid-like coordinate system we developed for the hexagon grid would be a 3 coordinate system representing the different vectors a point on the graph can take to reach another point. Such a system is shown in figure 6. This system provides a more natural and uniform way to calculate distances between points on the hexagon grid by

the use of 3 constant vectors. However, this system is less intuitive for our own spatial understanding of where two points lie relative to each other. For example, point (1,1,1) is 1 vertical unit above point (1, 0, 0). Ultimately we opted to use the more common square grid system as we valued having a more intuitive representation system for users despite the more complex logic needed for distance calculations internally.

The other piece of conversion required by the robot transitory frame is the conversion from camera board coordinate to robot board coordinate. The implementation for this

Figure 6. Alternative Board Coordinate Mapping



conversion utilizes the distribution of vertices/edges for each row to determine the specific row that a vertex/edge enumeration yields. We then calculate the vertex/edge offset in the row, if the specified vertex is the first vertex of the row, the offset is 0, while if it's the 3rd vertex of the row, its offset is 2. From there we use the knowledge of col number distributions for each row to determine the specific column number.

With these two conversion processes, we are now able to convert from camera board enumeration to the specified offset of where a piece should be. In our implementation, we are not saving any representation of the board, but rather doing the calculations everytime they are needed. Thus, no explicit preprocessing step is required for this frame. When executing the

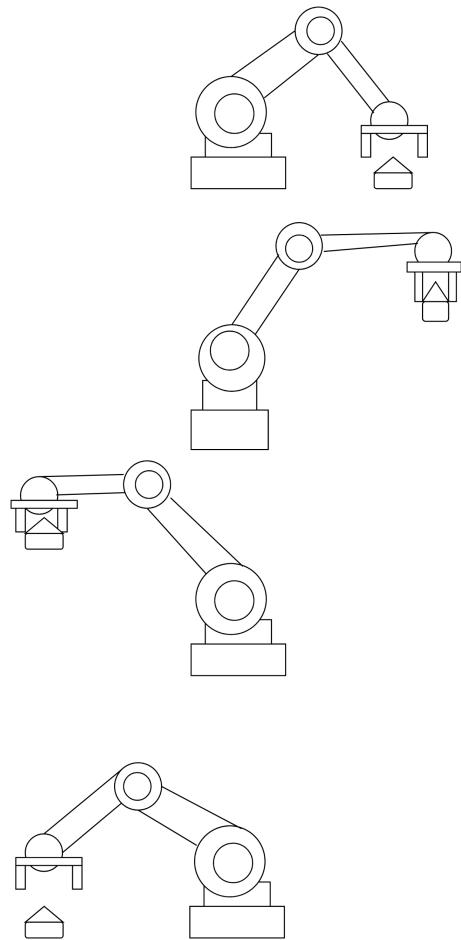
program and trying to find the offset of a specific board enumeration, we simply run the appropriate conversion functions.

Robot Arm Frame

One step we glossed over in the previous frame was how we determined the calibrated point connecting robot board coordinate with physical offsets. Since we have no sensor for us to detect this information for us, we decided to allow the user to do the calibration step themselves. At the bootup of the arm and start of our program we code a specific coordinate on the board and a best guess offset for that coordinate and ask for user input to make subtle changes on the positioning of the robot arm to reach that coordinate and confirm the positioning once they are satisfied. We then record this new position as the calibrated base point to base our calculations on for the rest of the board. This concludes the preprocessing board representation step of the robot arm frame.

Besides the coding of the calibration, we simply used the available api (which handled the inverse kinematics for us) of the robot arm to handle the grabbing of objects using the physical offset values from the arm. For every offset calculated, we run the procedure of hovering over the object, coming down towards the object and picking it up, hovering up again, hovering over the drop position, coming down towards the drop position and dropping the object, hovering up, then going back to sleep position. We add all of the hovering to ensure that nothing is knocked over on the board while doing all of the placing.

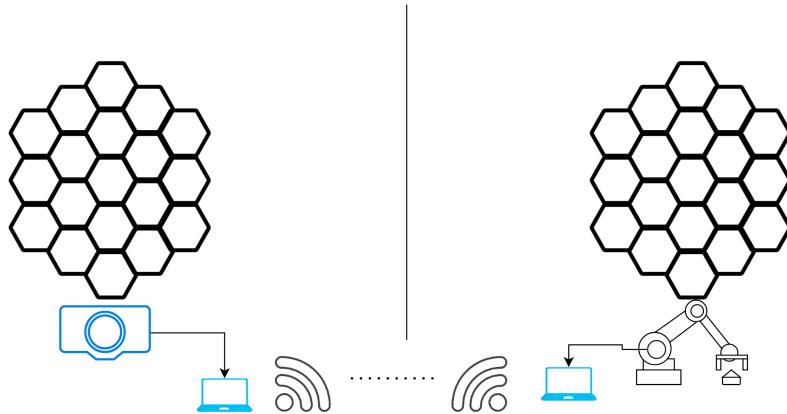
Figure 7. Robot Arm Pickup/Drop Procedure



Setup and Procedure

Our physical setup was as follows: one Catan board was fitted with an overhead USB webcam (no specific model requirements) connected to a laptop running on mac handling the cv code overlooking the grid of the board and another Catan board was set up with a Trossen robotics PincherX-100 robot arm connected to another laptop connected to ubuntu on the edge of the Catan board. The operating system of the laptop running the CV code should not matter, but the laptop running the robot arm code must be Linux. We ensured that the two boards had the same orientation relative to the camera and robot arm.

Figure 8. Physical Project Setup



Our software setup involved following the software setup detailed by trossen robotics in order to interface with the arm. Depending on the arm chosen, the arm id bash variable may need to be changed in `start_arm.sh`. Furthermore, the global variables for the arm type and best guess coordinates for the calibration code would need to be updated in `src/actuation.py`. Furthermore, a google cloud pub/sub account needed to be set up and both computers needed the proper authentication to access the proper channels. Thus, the global variables for the pub/sub ids need to be adjusted in both `src/actuation_module.py` and `src/detection_module.py`. Now, before running any scripts, make sure the catan board is setup with a circle counter blue side up on each hexagon. This is an important step that improves the accuracy of the opencv algorithm.

To run all of the code after all of the software and hardware has been set up: on the computer connected to the camera, simply run

```
$ python3 src/detection_module.py
```

If no video frame seems to pop up, or the video frame is all black, the `cv2.VideoCapture()` line may need to be updated to point to the correct usb port number connected to the webcam. After running the script, a few images should appear, showing the analysis of the board based on our algorithms. It is highly likely that upon first running the script, the results will be off and not quite the same as the results we got. This is due to the finicky nature of opencv where the light

conditions as well as the camera orientation play a big role into the ability of the opencv script to properly do its job. If the results of the analysis are unsatisfactory, make sure the camera is able to completely see the board and as directly overhead as possible. Furthermore, the threshold values of the code might need to be adjusted and played around with to fine tune the result. For reference, our setup was run on a workbench in the Yale CEID after the sun had set, thus our values are tuned towards that environment. If all goes well, the opencv script should now be working properly. At this point, the script should prompt the user to press space to take a picture of the board in its initial state (the photo may take a second to take, don't click the spacebar twice until after waiting a few seconds). Now place pieces on the board and click the spacebar. The pieces should be detected by the camera and a message should be sent to the cloud. (Note after taking the first picture of the board, try not to move anything on the board other than adding pieces, otherwise the difference detection algorithm may detect false positives). Now, we have completed the work that needs to be done by the camera.

To run the code for the computer connected to the robot arm, run:

```
$ ./start_arm.sh src/actuation_module.py
```

The start arm shell script will handle the initial setup of the arm and start the actuation_module.py script after the arm has been initialized. After running the actuation_module script or if the script is closed out, the arm does not need to be reinitialized with the start_arm.sh script (unless the new opened console running rviz is closed), rather the script can be run again by simply calling

```
$ python3 src/actuation_module.py
```

The first thing the actuation_module.py should prompt the user to do is to calibrate the arm to hover over a specific board coordinate. After calibration is complete, the robot will be ready to receive pub/sub messages from the google cloud. If at any point the arm doesn't complete a motion and outputs an error to the console, it's most likely that the arm was unable to reach the specified coordinate. Either the coordinates need to be readjusted, or the arm is simply

physically unable to reach the position and orientation specified (the pitch also affects the ability of the arm to reach a specified position). If all things go well, both the arm and robot should be connected and working now!

Further work

Our project succeeded in establishing a proof of concept of the main architecture required to detect a piece on a board and have a robot effectively take a piece and move the same piece to the exact same position on another board. But, to do so, we had only implemented a bare minimum set of rules and scenarios specific to the catan game. A number of further work would center around furthering the ability of our architecture to play a real game of catan. For one, we would want to expand the piece detection algorithm to also consider the color of the object to decide which player's piece to place. Furthermore, we would like to determine an algorithmic way to determine the type of piece placed on the board (whether it is a city, settlement, or road), we imagine this could be handled by either taking the relative sizes of each piece to act as a way to differentiate the pieces or to train a machine learning algorithm to detect the piece type. The machine learning approach can also be another approach to detecting the pieces on a board in a way that is not sensitive to slight perturbations on the board after taking the before picture for the difference detector.

Furthermore another feature that we have drafted up an algorithm for but have not fully implemented would be the edge detection and translation. The idea for implementing this feature would be to simply run a modified dfs on all detected vertices and to take the midpoint between the two vertices to determine the center of each edge of the catan board. After detecting all of the edges, we can then create edge enumerations and finish the translation from edge enumeration to board coordinate.

Another nice large improvement involves setting up the infrastructure for multiple players playing the game. Because of resource constraints, we only had one arm to work with and thus

could not create a multiplayer setup. We have our envisioned infrastructure for how we could create a system to support multiplayer as described earlier in the report, but we would like to actually implement the code required to handle turns, cohesion between the various cameras and arms, as well as deal with the potential issues that may arise as a result of the camera needing to also filter out the robotic arm when processing the board. It would also be nice to test transferring the computer vision and robot arm handling code to a raspberry pi or other smaller processing device so that there we can more fully integrate our system and avoid needing to have a computer for every single device for every single player.

Finally, one more nice to have feature that might improve the usability of our project would be to implement hand detection code. This would eliminate the need for having the user press the spacebar every time they wanted to make a move, more fully automating the system but also making the product more seamless and intuitive. The idea would be that when a player is taking their turn the camera would first take a picture of the initial state of the board. The camera would then wait until it detected a hand. Once it does, it waits until the hand is no longer detected by the camera / leaves the board and then processes the board looking for new pieces. If there are no new pieces, it keeps waiting, otherwise it sends the actions over to the robot and ends the turn.

Camera coordinates to real world coordinates:

<https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/>

Work Division

The work of the project was generally divided into camera/computer vision and robot arm/board representation problems with Brandon focusing on the former and Eric focusing on the latter. Pub/Sub communication work between the camera and robot arm was handled by Eric. General architecture setup was generally conceived by Brandon. Similarly, in this project

report, we each wrote about the specific components of the project we focused on. But, despite all the divisions of labor, we both dedicated time in helping each other solve and think through the various problems in both focuses of the project.

Appendix (Original Senior Project Proposal)

CPSC 490 Senior Project Proposal
Brandon Liu, Eric Chien
Advisor: Lin Zhong

Gammix: Mixed-Reality Multiplayer Board Games

Introduction / Motivation

Many of us have played games like chess online. Especially through the pandemic, my family and I frequently played an online version of board game Settlers of Catan together through the website colonist.io. These online versions are fun, but lack the tactile experience of physically moving game pieces and seeing the board and gameplay live in 3-D space. But what if you could still play with others online, but using the real-life, physical board game in a straightforward way? We propose designing and building a system called Gammix for such mixed-reality multiplayer board game experiences.

We are inspired by the system built by engineer Shane Wighton that allowed him to play pool against other players online, but on an actual pool table.⁶ It does so by projecting an interface onto the pool table, and by employing a custom built robotic pool cue that automatically lines up and makes the shot with the force and angle specified by the online player. While Wighton initially built the robotic pool cue and projector system as a way to guarantee sinking every shot via sophisticated software, the added benefit of enabling an online multiplayer experience on the physical table turned out to be the real highlight. The project left him wondering: “What about this game makes it so fun?” After seeing his system, I imagined a version of it but designed for and projected atop a Catan board, and I wondered the same thing: Would it be fun?

Project overview

To explore the question of what makes a mixed-reality game experience enjoyable, we propose a two part senior project. The first part consists of designing and building a minimally working prototype of Gammix, specifically to play checkers, on a physical board but against online players. We chose checkers as it is one of the simplest board games. This part will take place over the course of a month and a half, from mid-February to end March. Part one consists of refining the high level requirements detailed below into a full specification, designing each aspect of the overall system, then implementing them in software and hardware where needed. We plan to write our code in a public Github repository and release it open source. In the spirit of open source software, we will aim to architect our project in an extensible way, specifically for adding/modifying games, such that specifications for other popular games (like chess, or Settlers of Catan) may be added by ourselves or other contributors in the future in a relatively simple way. One example of such a consideration would be designing the software to be able to

⁶ <https://www.youtube.com/watch?v=vsTTXYxydOE>

handle an arbitrary number of players (within reason), even though checkers is only a two person game.

The second part of our senior project will consist of usability and user experience (UX) testing and interviews using our working prototype. This consists of formulating a full list of questions and a testing methodology, which we will create concurrently with our designs during part one and execute in part two. If needed, we may perform contextual inquiries and/or interviews with users playing board games in real life as well as their equivalents online to understand a general baseline experience. We would test our prototype in a location like the CEID on campus, separating the players in different rooms to simulate a distance. When designing our testing methodology, the overarching questions of “How easy is it to use and learn to use?” and “What can we learn from users’ experiences with this mixed reality game?” will guide us.

Deliverables

- Game board / Projected game interface
 - Add ArUco markers to detect specific features with computer vision code
 - This would include the corners of the board to enable accurate scaling and board state update
 - Also add to game specific pieces such as checkers and chess pieces (if needed) to be tracked by the camera relative to the board state
- Software / Phone app
 - Propose building an app using Flutter and the Dart programming language to allow for cross platform compatibility (iOS, Android), as well as built in access to the phone camera
 - Will connect over the internet to other players with the app so that the gameboards of both players will be synchronized to each other’s
 - Camera images will be processed using OpenCV ArUco detection code to state of the board, transmitting state updates to both players
 - Multiple games plan to be developed including checkers, chess, and potentially Catan
- Projector apparatus
 - A physical apparatus to mount the projector and phone, such that it can project the interface onto game board in real time as the game progresses
- Final written report
 - Detailing our design and implementation processes and outcomes with descriptions, diagrams, and/or photos
 - Sharing findings from our UX/usability testing

Part one estimated timeline / Subgoals

- Feb 21: Agree on a overall project specification/direction after meeting again with Prof. Zhong (0.5 week)

- Also acquire/purchase any equipment such as projectors/cables if needed
- Feb 28: Finish an engineering design document for the Flutter application considering the relevant trade-offs, and start development (1 week)
 - App should be able to detect ArUco markers and connect to the projector
- March 7: Construct a working projector apparatus to hold the phone and projector over the game board using materials in the CEID or Architecture lab (1 week)
- March 14: Finish working detection of the board, and rendering/scaling of the game board interface (1 week)
- Spring break
- March 28: Finish working code to detect game state and communicate it with a backend storing information about a game (1 week + spring break)

Additional Specifications Details

Game board / Projected game interface

The plan is to have a generic board of reasonable size with edges marked out with ArUco markers so that players with any sort of game board could theoretically integrate with our app without needing any specialized hardware (potentially, players could utilize any space of projectable size, without even a board). These ArUco markers delineate the playable space and would help initialize the play space. This initialization step will be programmed by us to analyze the area set by the ArUco markers by taking a picture with the camera and subsequently scaling the Gammix interface to the play board such that the projected interface is aligned with the actual board.

After initialization, our plan for implementing the rest of the board includes projecting a general grid to serve as the backbone of the analysis effort done by the camera component along with any other objects specified by the game. For the example of checkers or chess, after each user's move, the interface would project onto the board showing which their opponent had moved and to where, and then that player would update the board by picking up that piece and physically moving it to reflect that.

For relatively simple games, game pieces could be simply chips with ArUco markers attached to them with coding on the end of our detection algorithm to keep track of the location and state they are in relative to the master grid. For more complex games, more intricate methods of maintaining piece state info will likely need to be developed. In the case of checkers, with red and black interchangeable pieces, we anticipate simple color detection computer vision code will suffice.

Software / Phone app

The app serves as both the main software hub as well as interface for the user, actually rendering what will be projected onto the board. The idea for development of the app is to use Flutter, a cross platform development toolkit by Google, to build an app capable of connecting users across Wi-Fi into a Gammix game. The app would be responsible for synchronization of the game between two players in different locations. It will keep track of the local game state of

the board of the player, periodically sending and receiving data from the other player's phone in order to ensure that the board state is synchronized between actions from the two players.

Since the games we currently intend to implement will be strictly turn based in nature, we will design a way to ensure that there is an order to the moves made, perhaps utilizing explicit movements/conditions to end or start a turn or to signal information to players.

Furthermore, the app will access the phones' camera to keep track of the player's board, taking photos in time intervals to be analyzed in OpenCV with ArUco markers. We wish to efficiently utilize the powerful processing capabilities of smart phones to achieve this goal. As the user interacts with the piece and makes their move, we can either detect when a move has been made automatically by calculating differences in captured images over a period of time, or simply have the player press a button when their turn is over for the MVP. By designing a general enough setup, ideally applicable to many different board games, we hope to create a system that will allow easy extensibility to even more varying and complicated games with unique rulesets.

Other issues that will have to be addressed include common issues that arise from utilizing camera based sensing such as the issue of hands or other limbs blocking ArUco markers or other parts of the board that need to be tracked.

Other design considerations

Some consideration will also be given to the nature of the board for our board games including size and length and the possibility of including other hardware pieces to extend the sensing capabilities of the project. Other considerations include the best way to position the camera relative to the projector and if the camera from a phone is sufficient or if another separate camera would be needed (including bridging a communication method between the phone and camera).