MONASH University

FIT3077 Sprint 1 - (20%)

Team: CL_Monday06pm_Team377

1. Team information

a. Team photos and name

We have decided to name our team **KDQQ Interactive**, the name was formed from the initials of the four members first names (in Vietnamese) and the belief that we strive to provide new and fun interactive products.

Team photo:

Google Docs link:

https://docs.google.com/document/d/1rwewpUjpbnIenMir2Zw6BZ9E7zSbrWOmNuQLA7CyYTg/edit?usp=sharing

- This is where we collaborated on this document, and it also shows logs of member's activity and version history.

Google Drive link:

https://drive.google.com/drive/u/1/folders/1-GasZsvj13MoEW1VVdiBZD1CQIawce0K?fbclid=IwAR1vZZ9765ZnkYNXzvKKHC0xXpXkhtrnZS_7akVNl6ggiXONZL174Rnwx1I

- This is the Google Drive link for our drafts and other files used to make this document. Please see photos of our LoFi design and the Draw.IO file of our Domain Model here.

b. Team membership

i. Pham Nhat Quang Nguyen

Contact details: pngu0054@student.monash.edu, nguyenjoe0201@gmail.com
Technical and professional strengths: Python, JavaScript, React, PyTorch, TensorFlow, Robot Operating System.
Fun fact: Curious about AI and Robotics

ii. Nguyen Khang Huynh

Contact details: nhuy0018@student.monash.edu
Technical and professional strengths: JavaScript, HTML, CSS, Python, Java
Fun fact: A big fan of Front End, UX/UI

iii. Nhat Quang Nguyen

Contact details: nngu0112@student.monash.edu
Technical and professional strengths: Full-stack development, Mobile development, and Data Structures and Algorithms
Fun fact: An ex-MCAV member and majoring in Advanced Computer Science, I love both low-level programming and the more theoretical side of Computer Science!

iv. Tran Ngoc Duy Ngo

Contact details: ngotnduy@gmail.com (personal email), tngo0031@student.monash.edu
Technical and professional strengths: Web development (JS), frameworks, libraries such as Reactjs and tailwind, discrete mathematics.
Fun fact: hate data structure and algorithms, especially FIT2004

c. Team Schedule

In regards to the team's meetings, as the schedule and availability of each member doesn't align well with each other, we have decided to collaborate asynchronously online via a shared google docs for sprint one. To keep everyone on the same page, we have all agreed to discuss and explain any changes planned on our Facebook Messenger chat, and we will only carry out the change once everyone understands the issue and approves the changes.

Due to the previously mentioned schedule mismatch, we have only found time to hold a weekly stand-up at 1 PM every Saturday afternoon. During this time we will go review the changes we have made and discuss and allocate the tasks needed to be done for the next week. Other than this standup, we have agreed to stay after the Monday workshop to discuss any questions we may have with tutors and work out any problems we may have during the previous standup in person.

For sprint 1, our tasks only consist of design and documentation and not any programming yet; hence, our method of tracking member productivity is to use the version history and activity dashboard in Google Docs. During the standup, we can review each member's performance and discuss actions to take if a member is lacking contribution. From the next sprint onwards, when more programming-related tasks are introduced, we will discuss together how to allocate responsibility based on our interests, strengths, and weaknesses.

d. Technology stack and justification

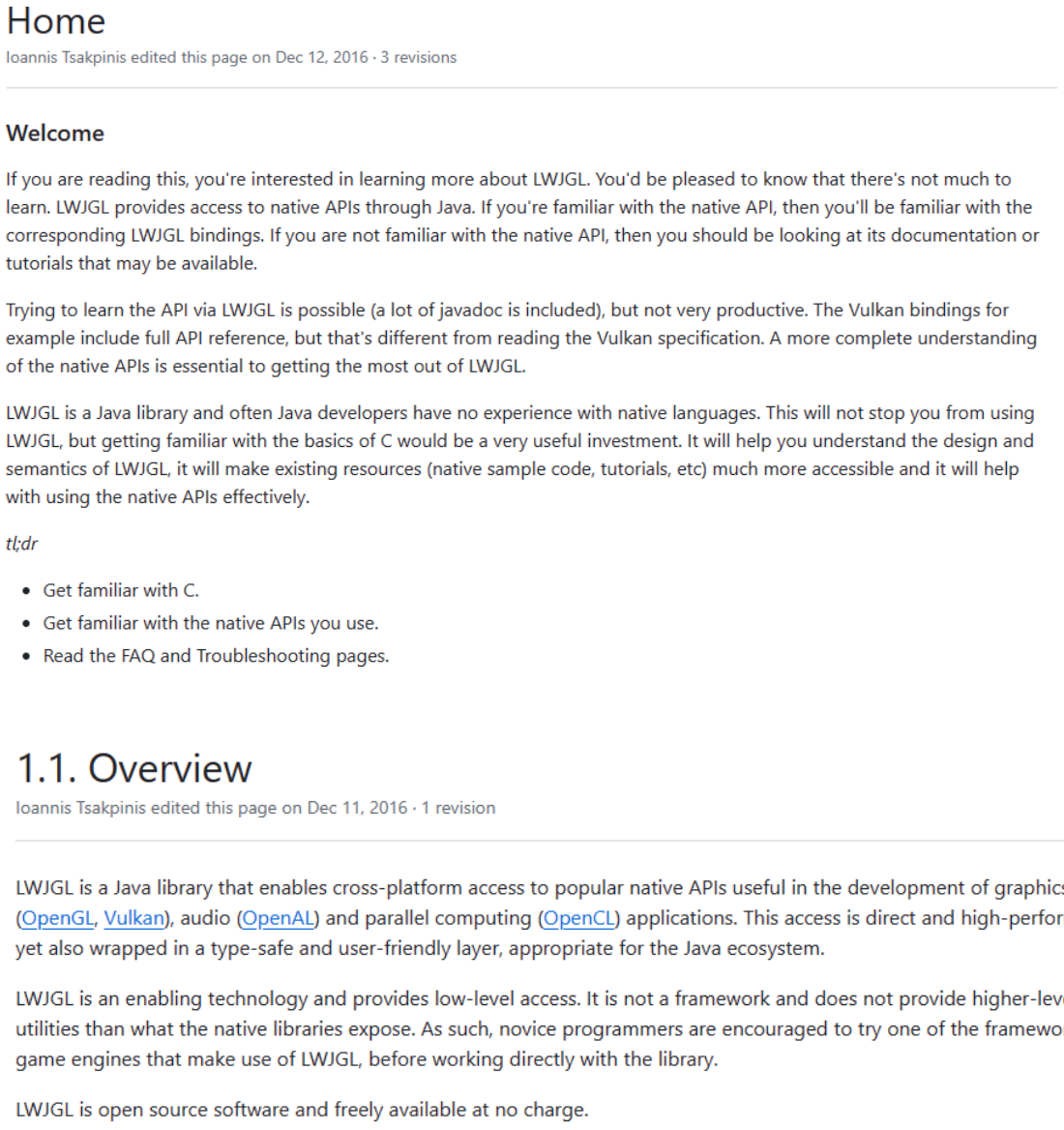
i. Programming Language

We were given the choice between Python and Java, we saw Java as the better fit for our team for the following reasons:

- FIT2099 was taught in Java, so we can transfer our experience more easily without needing to translate it into Python
- Java is statically typed and is better suited for Object Oriented Design than Python, for a team that haven't had a history together and relies heavily on models and documentation as the basis for collaboration, Java's strictness might allow us to avoid collaboration issues.
- Performance was not a major concern for us as this project is relatively simple. But Java wins in this department as well
- Framework choice: Some Java frameworks work really closely with lower-level C libraries. This would be a good opportunity to better understand what is running under the hood.
- The main game engine available for Python was PyGame, and with its main audience being hobbyists or beginners, we feel like we might encounter issues with support.

ii. Framework Choice

With Java being our language of choice, we wanted to find popular frameworks as they are usually still under maintenance and up-to-date, offering good and detailed community support and documentation. Our search leads us to two main contenders: LWJGL and libGDX. Both of these engines are good for small-to-medium game projects and offer good control and customization ability.



Home

Ioannis Tsakpinis edited this page on Dec 12, 2016 · 3 revisions

Welcome

If you are reading this, you're interested in learning more about LWJGL. You'd be pleased to know that there's not much to learn. LWJGL provides access to native APIs through Java. If you're familiar with the native API, then you'll be familiar with the corresponding LWJGL bindings. If you are not familiar with the native API, then you should be looking at its documentation or tutorials that may be available.

Trying to learn the API via LWJGL is possible (a lot of javadoc is included), but not very productive. The Vulkan bindings for example include full API reference, but that's different from reading the Vulkan specification. A more complete understanding of the native APIs is essential to getting the most out of LWJGL.

LWJGL is a Java library and often Java developers have no experience with native languages. This will not stop you from using LWJGL, but getting familiar with the basics of C would be a very useful investment. It will help you understand the design and semantics of LWJGL, it will make existing resources (native sample code, tutorials, etc) much more accessible and it will help with using the native APIs effectively.

tl;dr

- Get familiar with C.
- Get familiar with the native APIs you use.
- Read the FAQ and Troubleshooting pages.

1.1. Overview

Ioannis Tsakpinis edited this page on Dec 11, 2016 · 1 revision

LWJGL is a Java library that enables cross-platform access to popular native APIs useful in the development of graphics (OpenGL, Vulkan), audio (OpenAL) and parallel computing (OpenCL) applications. This access is direct and high-performance, yet also wrapped in a type-safe and user-friendly layer, appropriate for the Java ecosystem.

LWJGL is an enabling technology and provides low-level access. It is not a framework and does not provide higher-level utilities than what the native libraries expose. As such, novice programmers are encouraged to try one of the frameworks or game engines that make use of LWJGL, before working directly with the library.

LWJGL is open source software and freely available at no charge.

It is evident that the LWJGL library requires a steep learning curve before the team can actually take advantage of the features of it. Given that time frame of the project, it is not feasible to get familiar with such a complex and low-level library at this point of time. Hence, we have opted for the libGDX library which offers more specifically tailored features for game development.

In addition, the jMonkeyEngine framework was also considered as a potential choice. However, jMonkeyEngine is specialised in 3D game development. Thus, the game development process will be longer compared to a

2D version, and the bundle will require more hardware power to run, which is not an ideal choice since we want to showcase the game to students and parents.

Having said that, the libGDX library is a rather new framework to the team. Hence this is something that we might require some support from the teaching team during the development process.

2. User stories

a. Epic: As a player's piece, I want to represent the player in the game so that the player is engaged with it.

- As a player's piece, I want to be able to get out of the caves so that my piece can join the race.
- As a player's piece, I want to have a unique appearance so that I'm easily identifiable by a human.
- As a player's piece, I want to move clockwise the number of steps as labelled on the Chit Card I selected if it matches with the Volcano Card my piece is standing on so that I can progress in the game
- As a player's piece, I want my piece to get into the caves after completing the race so that I can win the game.
- As a player's piece, I want my turn to end if other pieces (or Players) are blocking the way so that I cannot move across the player and break the rules.
- As a player's piece, I want my piece to move the number of steps anticlockwise as labelled on a Chit Card with a Pirate Dragon on it so that I abide by the game's rules.
- As a player's piece, I want to be placed in a cave at the start of the game, so that I can start the game fair and by the rules.
- As a player's piece, I want to have exactly 4 pieces, one for each player so that there are enough Volcano Cards for each player to play.
- As a player's piece, I want my turn to continue if my selected Chit Card matches with the Volcano Card I'm on so that I'm rewarded for memorising the Chit Card, as per the game's rule.
- As a player's piece, I want my turn to be ended if the number of moves on the Chit Card I selected is greater than the number of moves needed to get back to my cave so that I play by the game's rules.
- (Extension) As a player's piece, I want a bot to play the game if there are fewer than 4 participants so that people can enjoy the game even if there aren't enough players.

b. Epic: As a cave, I want to be at the end and the start of the game, so that there is a goal for the player to work towards.

- As a cave, I want to let my players in and win the game only after they have gone around the whole board so that the game can be finished.
- As a cave, I want only to allow the piece that started on me to enter after they went around the board so that the game is fair and I allow the correct player to win.
- As a cave, I want to have exactly four caves on the board so that the game rules are abid.

c. Epic: As a Chit Card, I want to make the player memorize my content so that the game is more challenging and interesting.

- As a chit card, I want to have exactly 16 chit cards on the board so that the game rules are abid.
- As a chit card, I want to be randomly scrambled at the beginning of the game so that it will maintain the balance of the game.
- As a chit card, I want to be hidden at the start of every turn, so that no one knows my type and follows the game rule.
- As a chit card, I want to reveal my type when selected by a player, so that the player can see and remember my type.
- As a chit card, I want to stay revealed if I was flipped during each turn so that the player will know which Chit Card has been selected (following the game's rule).
- As a chit card, I want the player to select me only if it is their turn so that the game can progress fairly.
- (Extension) As a chit card, I want myself to be rescrambled after every turn so that it can minimise cheating and increase difficulty in this game (one of the user stories for Sprint 4).

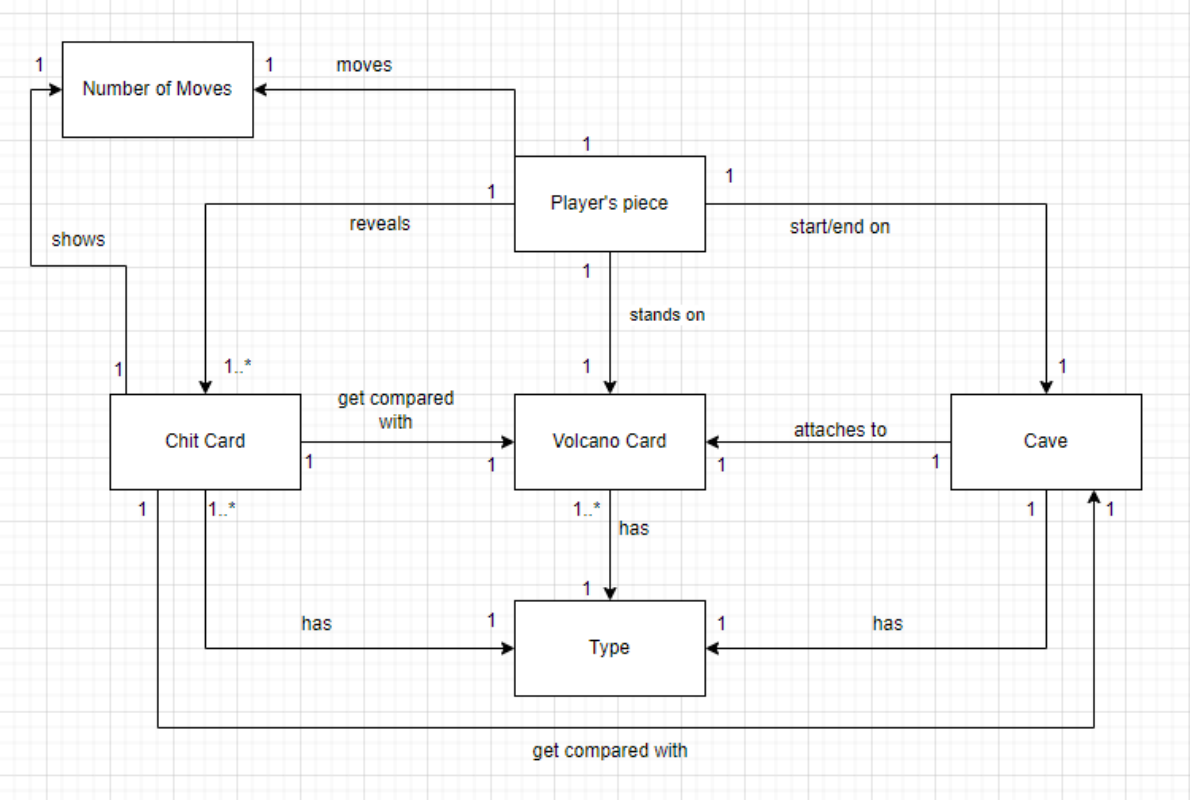
d. Epic: As a Volcano Card, I want to represent the path the player takes, so that there is a way for play to progress

- As a volcano card, I want to let the player know where they are on the board, so they know how far they have progressed.

- As a volcano card, I want to arrange it with the other volcano card in a closed loop so the player knows they can return to their cave and complete the game.
- As a volcano card, I want to be arranged with the other volcano cards so that there are a total of 24 sections for the player to move around in so that I abide by the game's rules.

3. Domain model

a. Domain Model Diagram



b. Domain Model Rationale

i. Entities

The first four entities we came up with are modelled from the physical pieces from the game, the Player's Piece, Chit Cards, Volcano Cards, and Cave Cards. However, there are differences between our interpretation of the model and the physical game elements. In the physical game, each volcano card has 3 spaces each with different types that the player can stand on, but this interpretation can make it hard to model the relationship with other entities. **So, our solution was to interpret each volcano card as a singular space that has a type that the player can stand on (instead of 3 like the physical version)**. Please keep note of this, as it is crucial for understanding our other rationales. **Another thing to keep in mind is that even though there is an entity labelled "Player's piece", it**

encapsulates all concepts relating to the player's interaction with the game, not just their representation on the board.

Aside from the 4 user roles derived from the user stories we found that the addition of 2 extra entities help better represent the flow of the game, and substantially aid in showing the relationship between some entities.

Type: this represents the concept of different types that the card in the game could take i.e. Baby Dragon, Salamander, etc. Chit Cards, Volcanoes, and Caves all have them, and comparing these types is one of the game's core mechanics. Without the Type entity, it is harder to understand how the three cards are connected. (NOTE: The Type here doesn't mean a variable class in programming, it just means what kind of 'animal' is printed on the cards).

Numbers of moves: This entity represents the concept of behaviour and interactions relating to the movement of the player's piece. A key aspect of the game is the number of moves allowed by a Chit Card when selected, there are multiple behaviours that can result from this depending on the state of the game (i.e End turn if the destination is blocked, win the game if the player's piece enters the cave, etc). We found that this responsibility to manage these behaviours doesn't belong to either Chit Cards or the Player's Piece, so we added the "Number of moves" entity as a way to encapsulate the processing of the movement behaviour of the player's piece (further analysis is available in the discarded alternative where we directly connect the Chit Card and Player's piece)

ii. Relationships

The relationship between the Player's piece and Chit Cards, Volcano Cards, and Cave Cards reflects the rules of the game. The player's piece starts out in a cave and also enters their cave to win the game, a cave can only 'host' one player piece, therefore the cardinality between the player's piece and Cave Card is 1 to 1. The player's piece progresses by moving on the Volcano Cards, and one Volcano Card only lets one player piece on it, and a player's piece can only be on one Volcano Card at a time; therefore the cardinality between Volcano Card and the Player's piece is 1 to 1. The player can reveal at least one Chit Card in their turn, but they can reveal more if certain conditions are met, so the relationship between the player's piece and Chit Card is 1 player's piece to 1 or many Chit Cards.

The comparison of the player's selected Chit Card and the Volcano Card that their piece is standing on is a core aspect of the game. Therefore there must be a relationship describing this comparison between them. Since only one Chit Card gets compared with one Volcano Card at a time, we believe that the one-to-one cardinality suits this relationship the best. Chit Cards also compare with Cave Card to get the player's piece out from their starting position, they have a one-to-one cardinality relationship between them by the same logic.

By the rulebook, a Cave Card must be connected to a Volcano Card via an indent on the Volcano Card, this is to enable the player to start the race by moving from their starting point, being their respective Cave Card, onto the Volcano Cards. There has to be a relationship between the Cave Card and Volcano Card to describe this functionality of the game, and because only 1 Cave Card is

connected to 1 Volcano Card, the best cardinality to have for this relationship is one-to-one

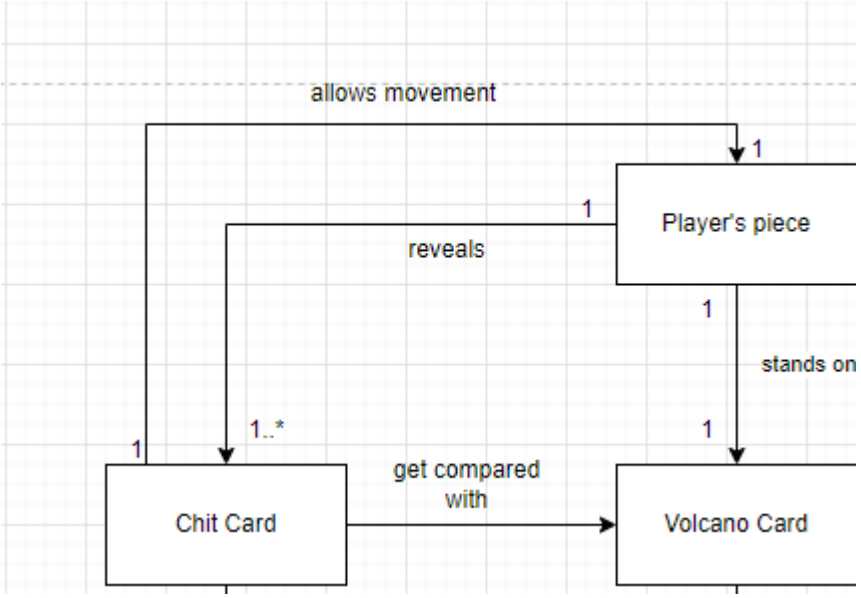
Chit Cards, Volcano Cards, and Cave Cards each have a common type being Salamander, Baby Dragon, Spider, and Bat (Chit Cards can have the Pirate Dragon type as well). For a Chit Card, a card can only have 1 type, but there can be multiple Chit Cards of the same type (i.e the 3 Salamander Chit Card vs the 1 Salamander Chit Card), so the cardinalities for the relationship between Chit Card and Type is one Type to many Chit Card. Similarly, the same logic applies to Volcano Cards, since there are 24 Volcano Card sections but each can only be one of four types, so the cardinality is 1 Type to many Volcano Cards. There are only 4 caves, and each cave has to be of a different type, so the cardinality between Cave Card and Type is 1 Type to many Cave Cards.

In addition to its type, Chit Card also shows the players how many moves to take, the number of moves on a Chip Card doesn't change and is an 'intrinsic attribute' (not in a programming sense) of the card. Therefore the cardinality that best suits the relationship between Chit Card and Number of Moves is 1 to 1.

Every time the player selects a Chit Card, they can move their piece by 0 or more moves depending on the state of the game (0 because the chit card doesn't match or the destination is blocked). There has to be a relationship between the Player's piece and the Number of Moves to describe this game's functionality. Since the player moves none or multiple times in their turn, and throughout the entire duration of the game, the best cardinality for this relationship is 1 Player to 0-many Number of Moves.

c. Discarded Alternatives and Justification

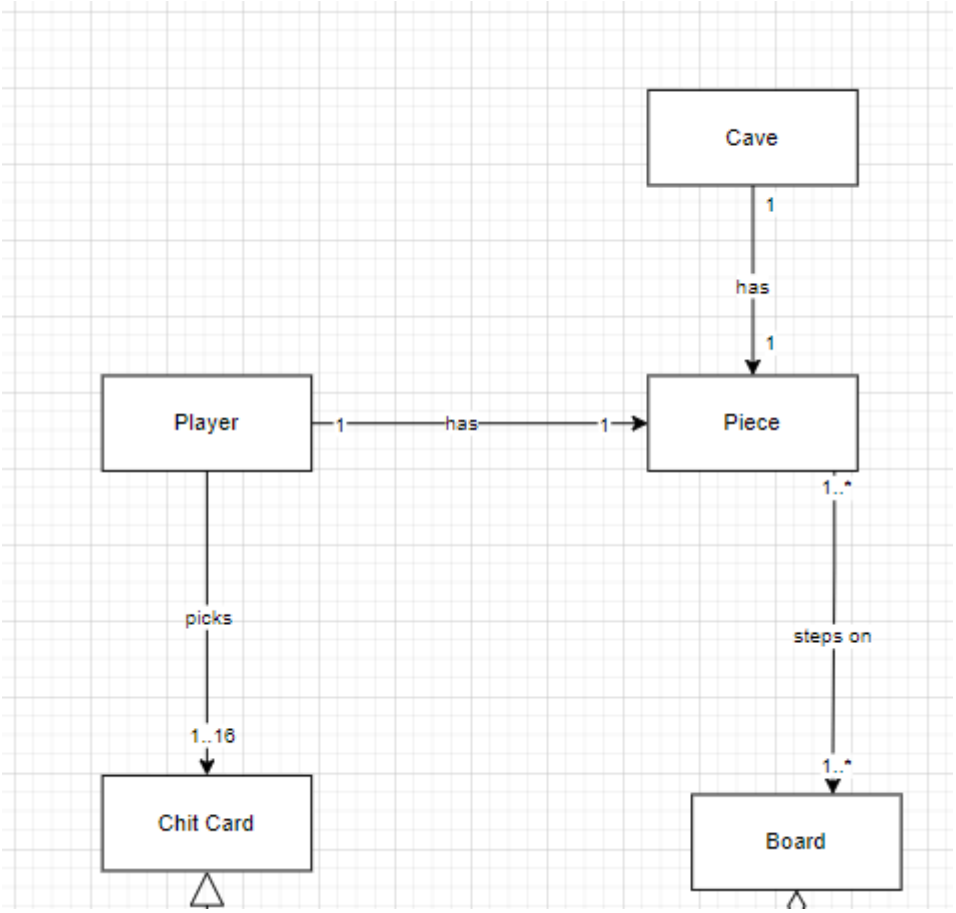
i. Relationship between player's piece and Chit Card



The reason we came up with this during our initial brainstorming was because it seems intuitive that the action of flipping chit cards causes the player piece to move. However, after further analysis, this seems to be an oversimplification due to the following reasons:

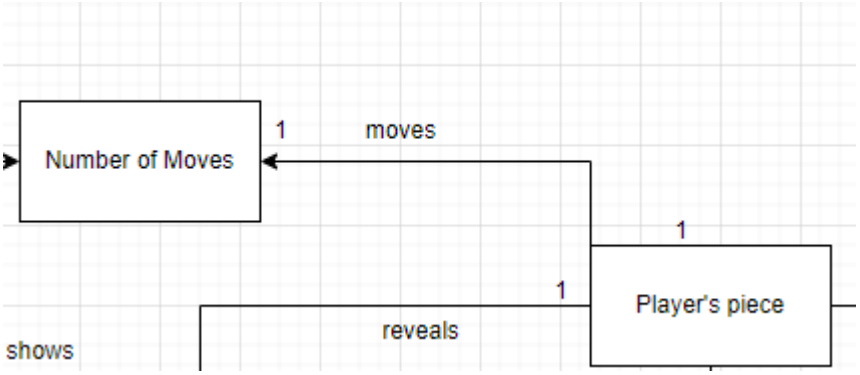
- Other game states influence the movement of the players i.e other pieces blocking or being unable to move past their cave at the end of the game
- Complex relationship - there are too many things to consider when considering the cardinality of this relationship, a Chit card can be flipped by many players, a Chit card can be flipped by a single player multiple times a turn, a Chit card can not be flipped at all, a player can flip many chit card, etc. We could've used a many-to-many cardinality, but that is both bad practice, and it doesn't aid in explaining our thinking, which is the point of the domain model.
- Cyclic pattern - You can see there is a cyclic relationship between the player's piece and the chit card. Despite not being a UML diagram, we fear that this could be a bad practice and will impair our implementation design down the line.

ii. Separate player entity



This idea was scrapped as we found it to distinguish the concepts that these two concepts encapsulate. Having two entities for the same purpose of modeling the player is both ambiguous and redundant. Merging the two entities also makes it easier to model the relationship with other entities in the game, but we found it difficult to find a name that would fit and flow well with other keywords in our model.

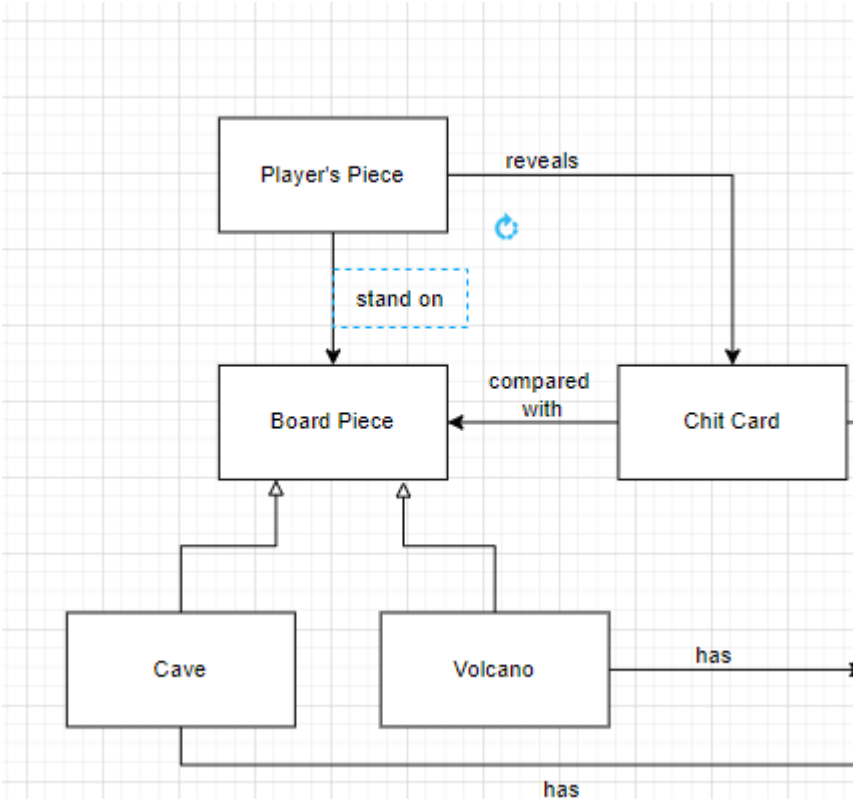
iii. 1 to 1 player's piece to a number of moves cardinality



There was a dispute within our team on the cardinality of this relationship. The argument for this was that a player could only move once per chit card flipped. But after thorough discussion, we realized it is better to use the context of the entire game, rather than a player's action when describing this relationship. What we meant by that was, that the player can move multiple times in a game, and the

Number of Moves entity represents the movement in the entire game, not just after flipping a chit card. Due to this counter-argument, the whole team agreed that our current version is better.

iv. Board Piece Generalization for Volcano and Cave Cards

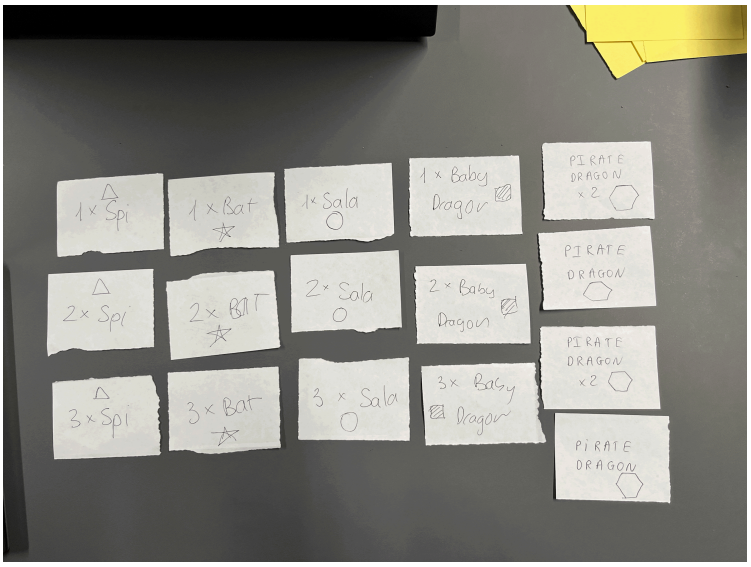


The generalization comes from the idea that the Volcano and Cave cards have a lot of the same functionality: Being the stepping piece for the Player's piece, having a type, and being compared with the Chit Card. However, we realized that our rationalization for this entity's existence relies too heavily on implementation reasoning. Conceptually, it is not distinct, nor does it help explain the other entities that it generalizes. In conclusion, the Board Piece entity was removed because it was not necessary.

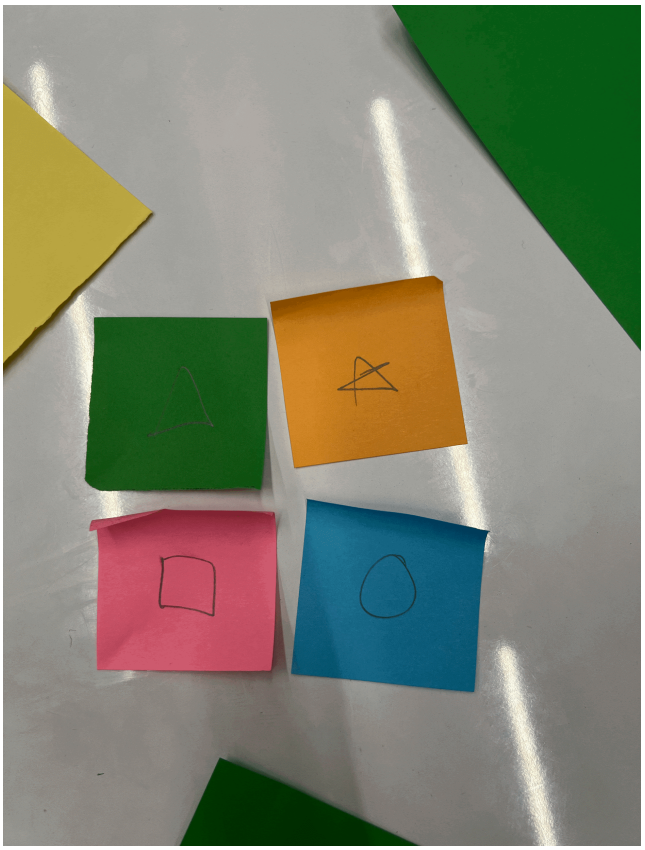
4. Basic UI Design - Lofi

1. Piece type

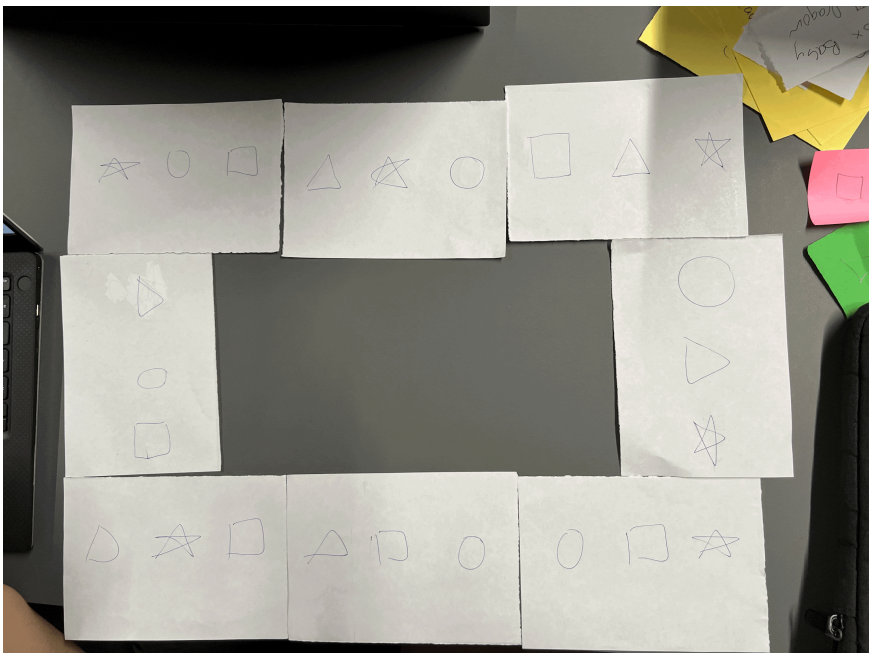
- 16 chit cards

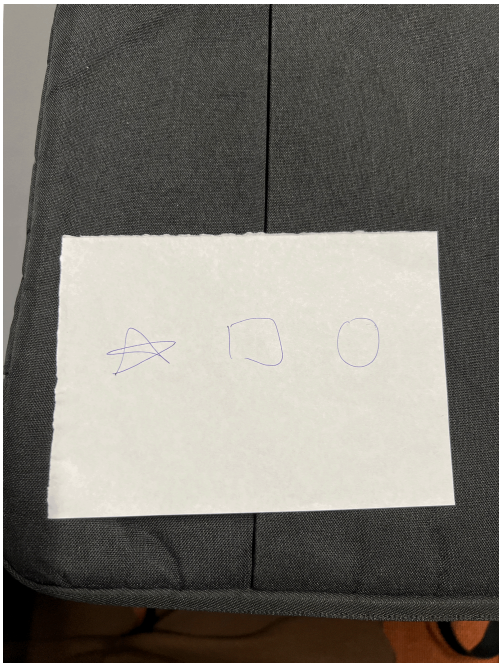


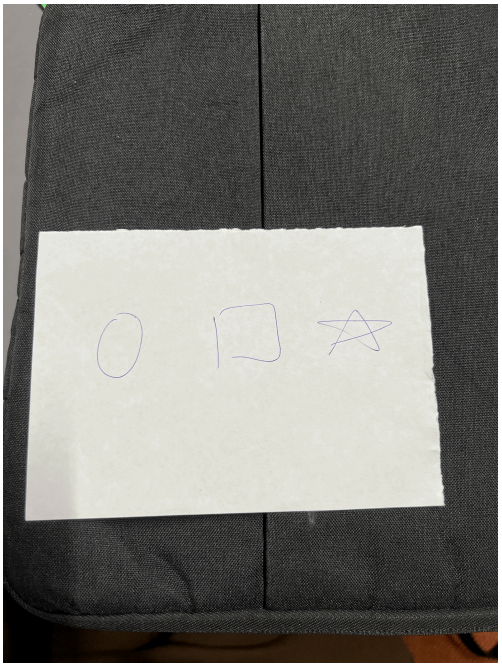
- 4 player pieces



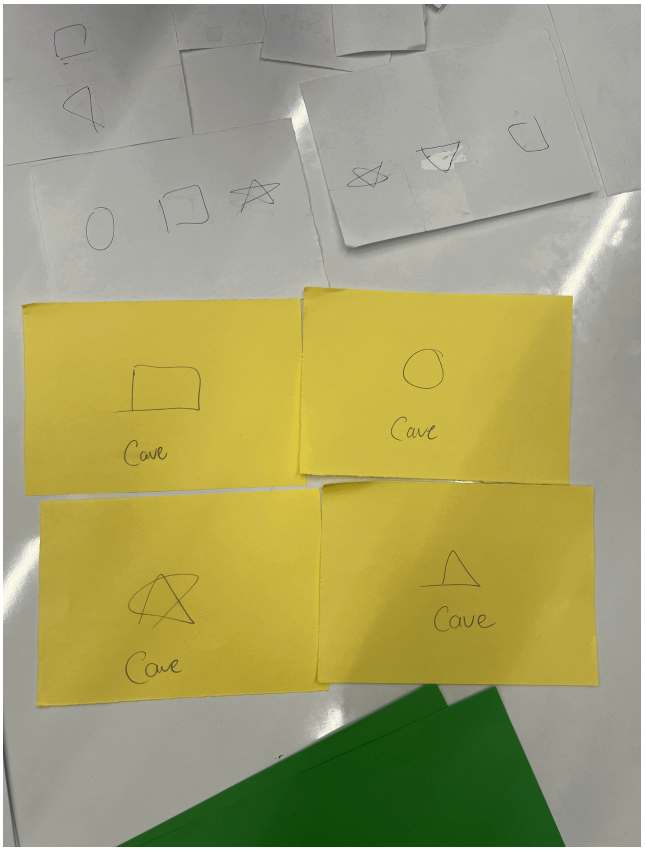
- The game board



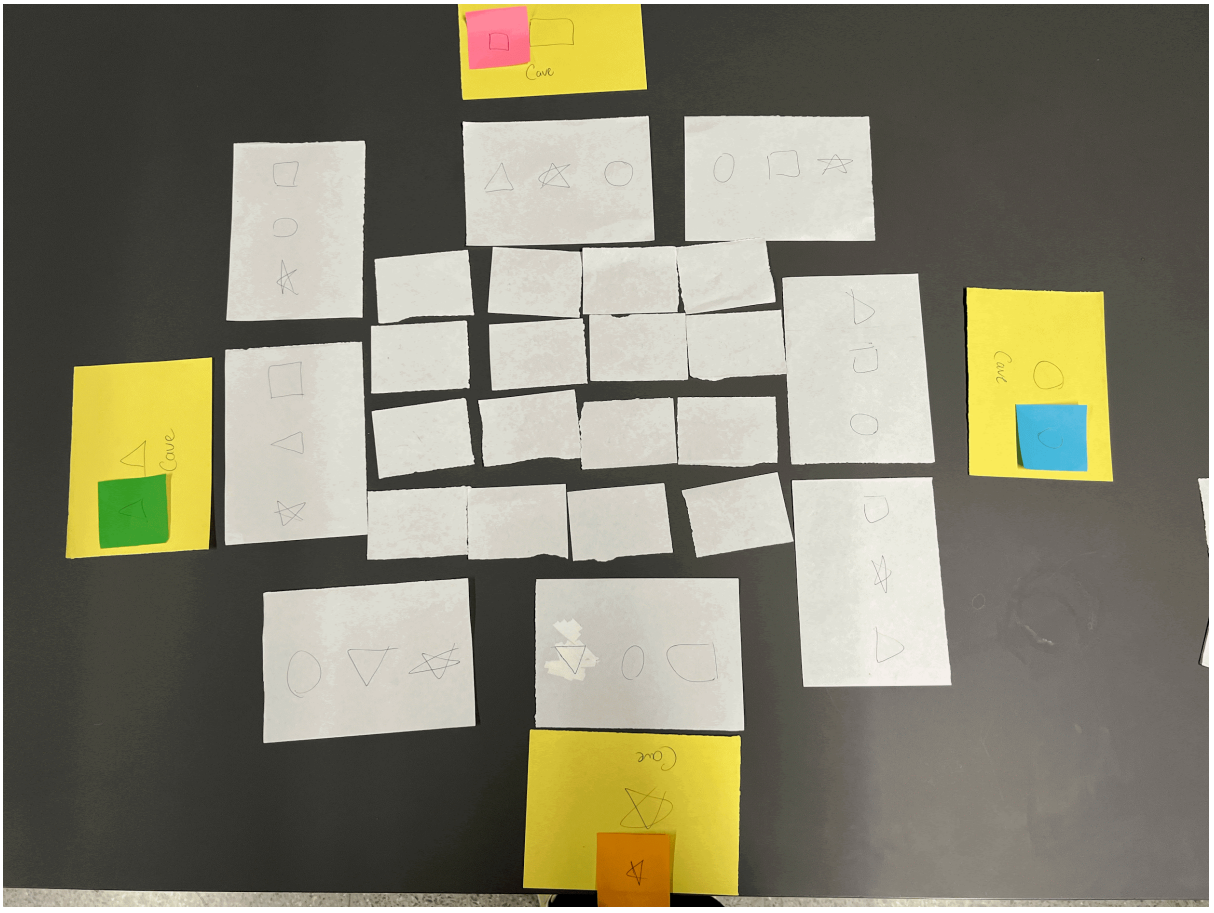




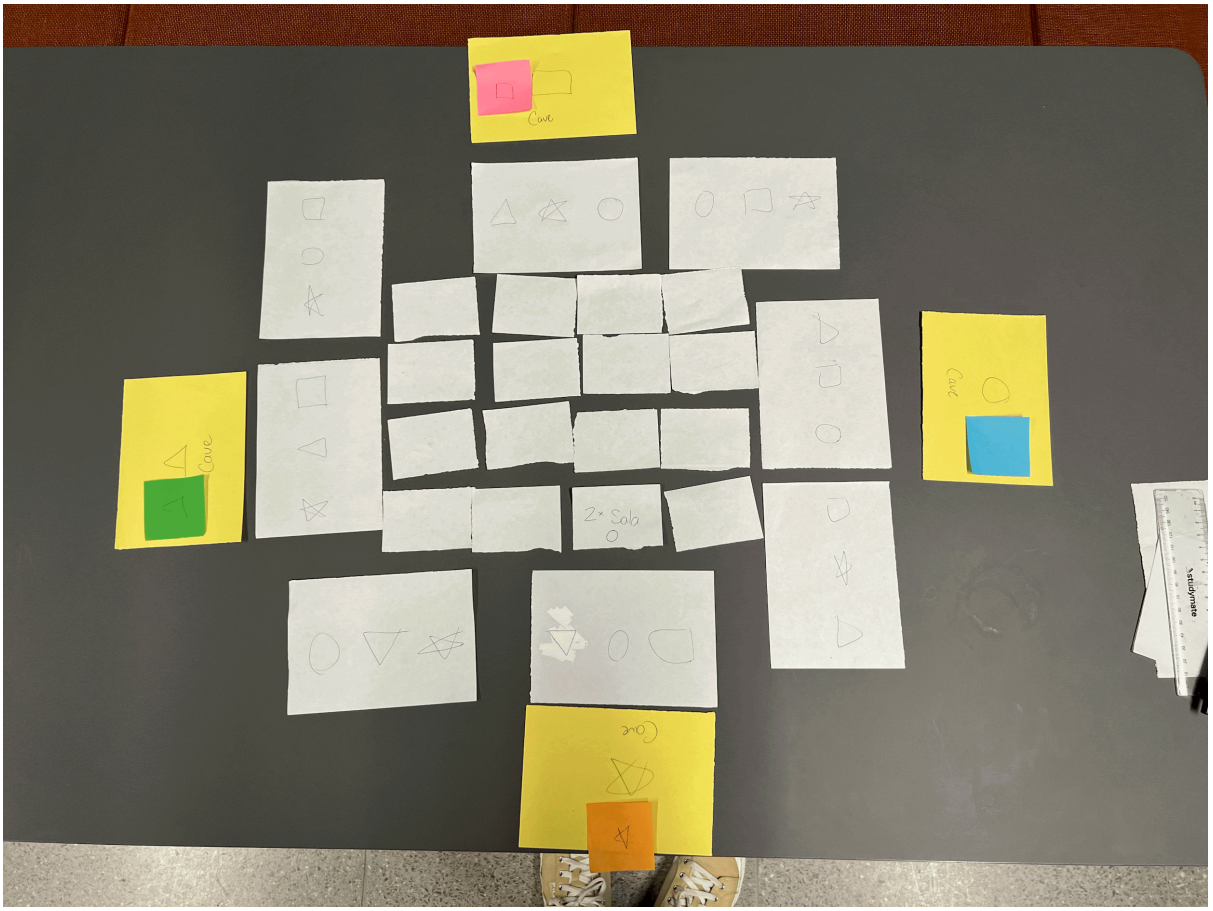
- 4 caves

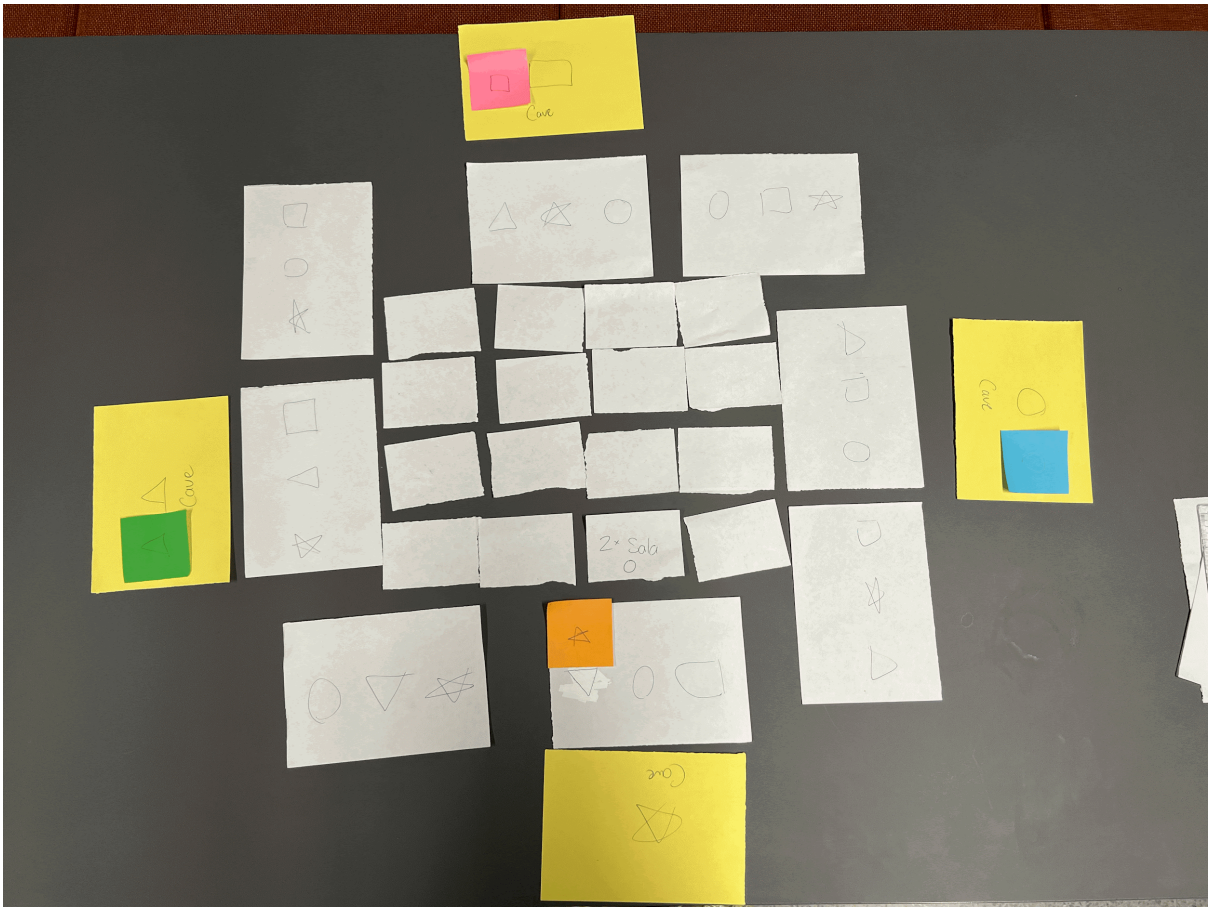


- The initial stage of the game

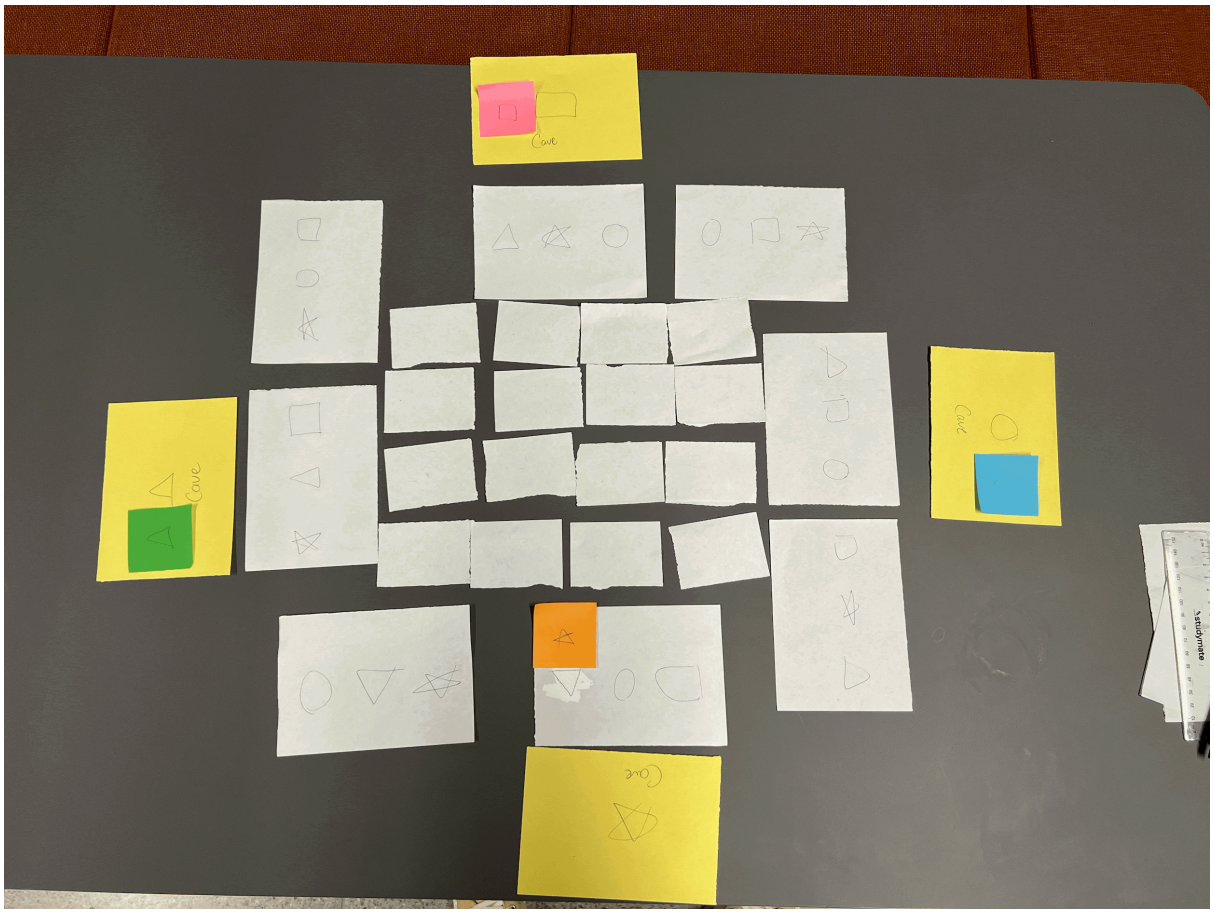


- At the beginning of the game, the star player picks the chit card which is the 2 circles. The star pieces will move 2 steps clockwise.



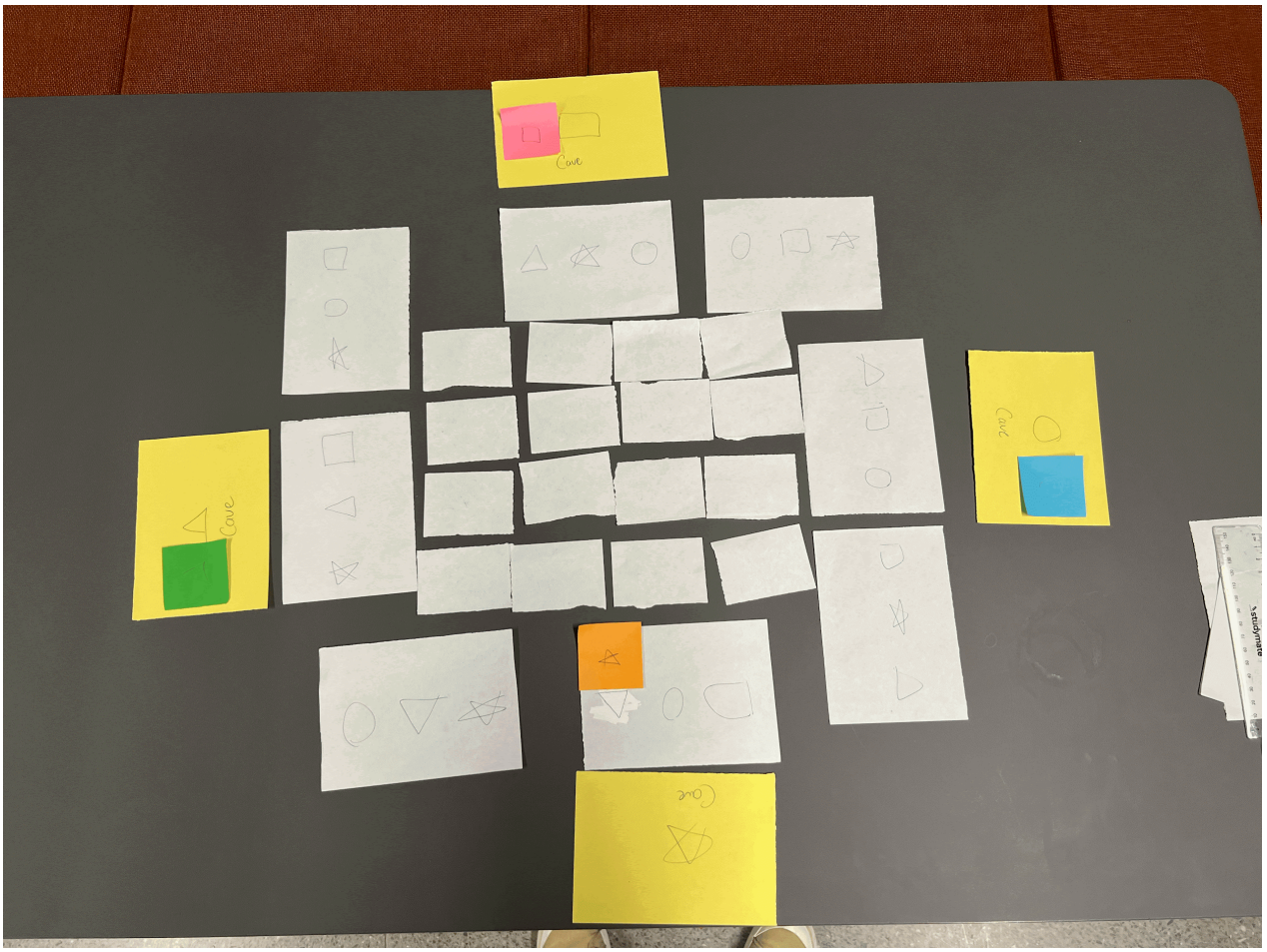


- The chit card then returns to its initial stage, and the turn ends.

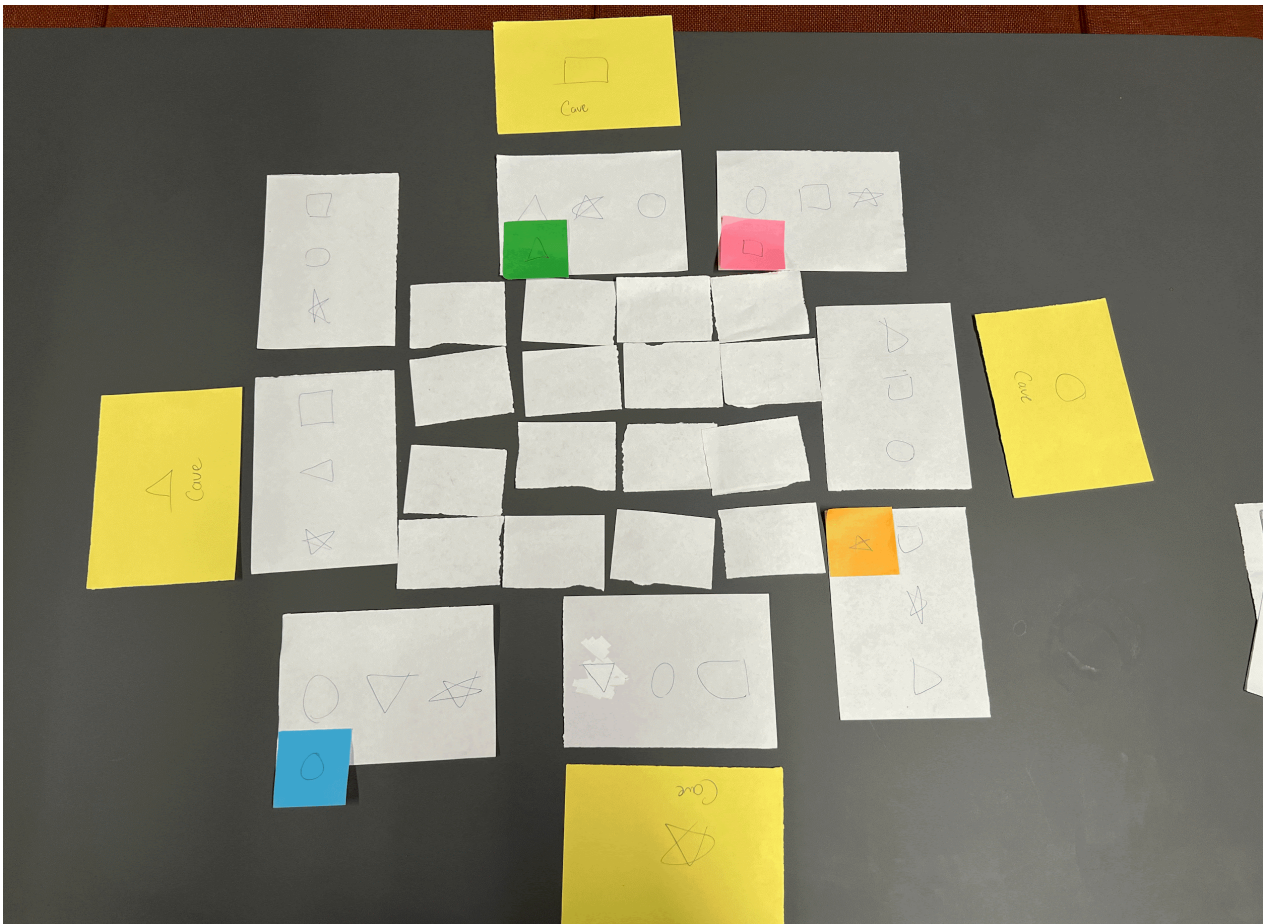


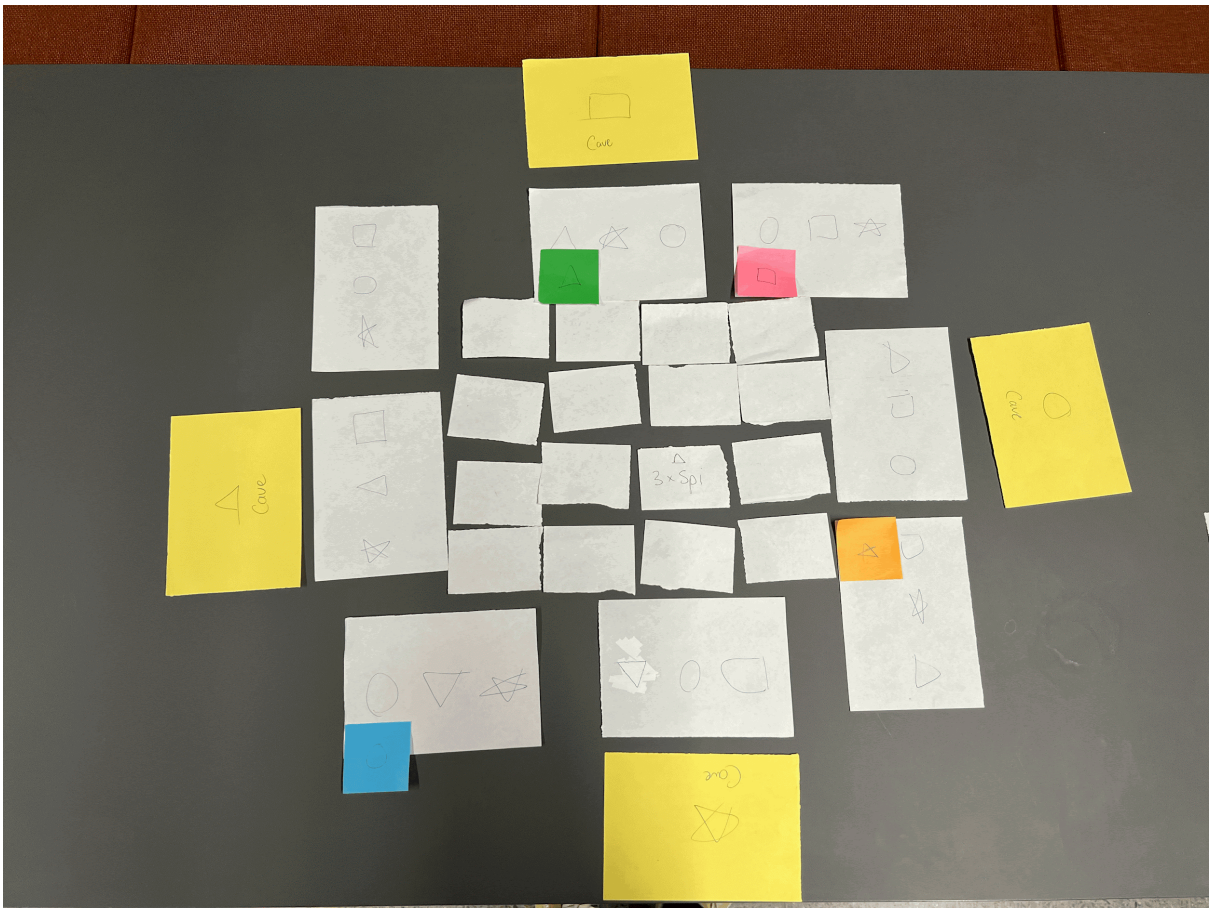
- Next, the green triangle player picks up the chit card which is the hexagon, and does not match the triangle on the volcano card and doesn't move.

- The chit card then returns to its initial stage, and the turn ends.

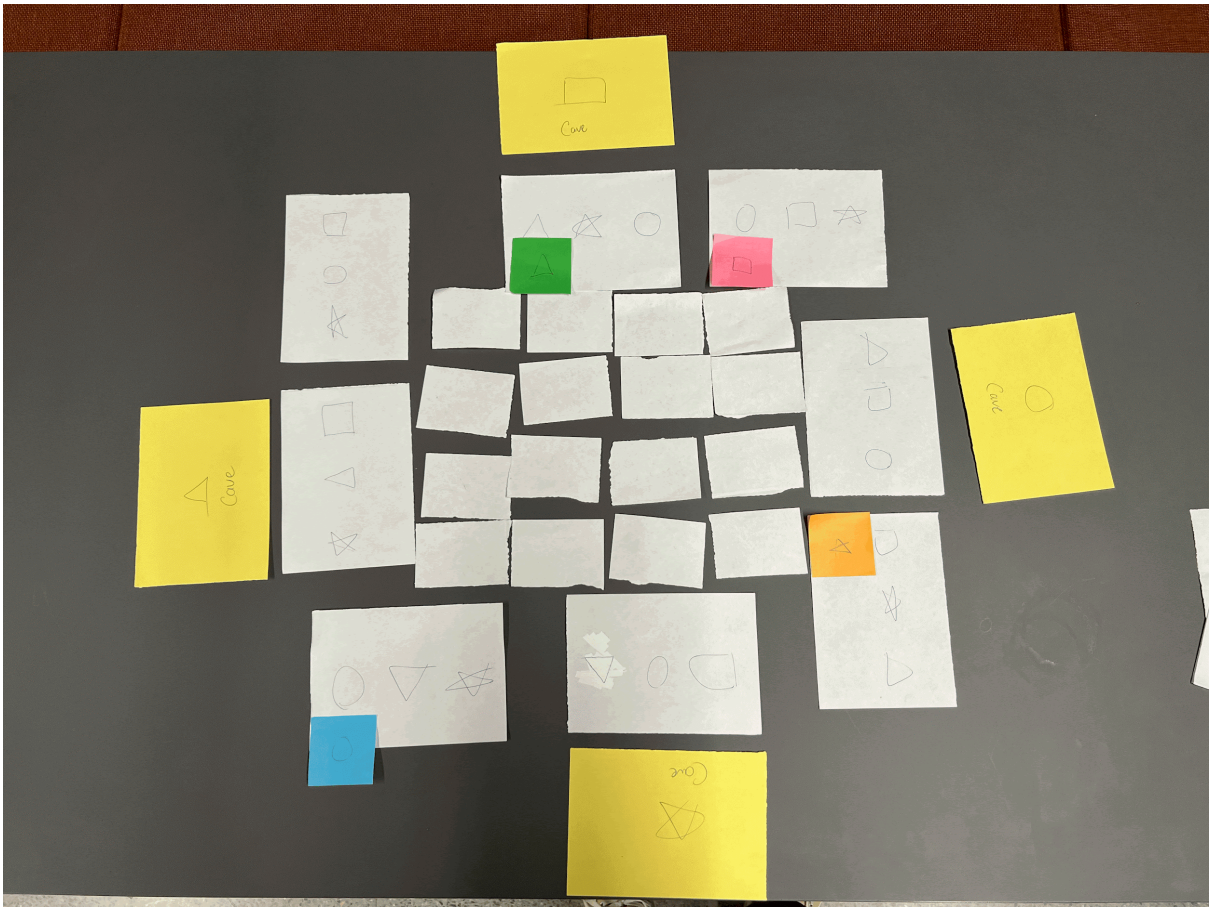


- The green triangle player picks up the chit card which is the triangle with 3. It does not move since there is a square piece block in the way.

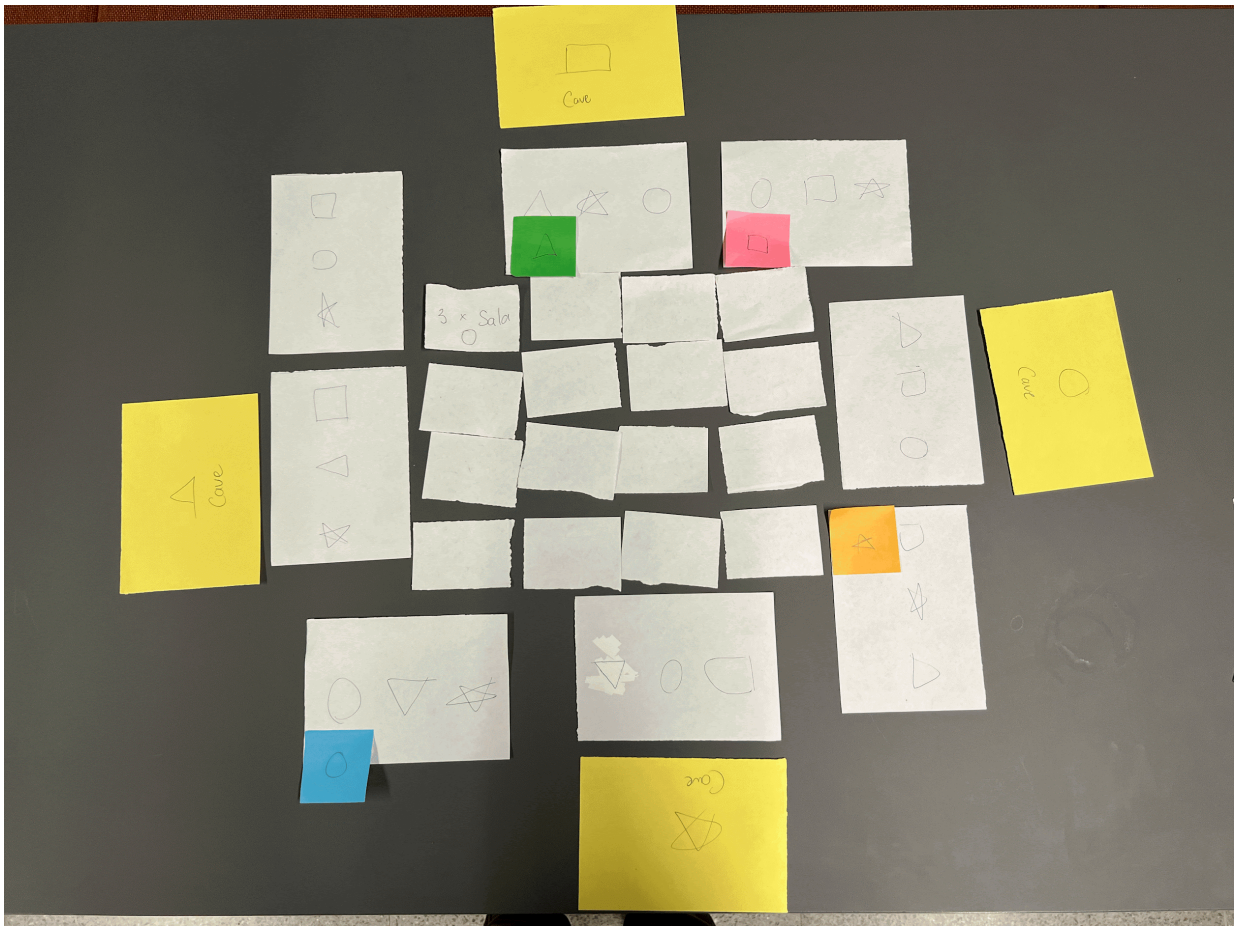


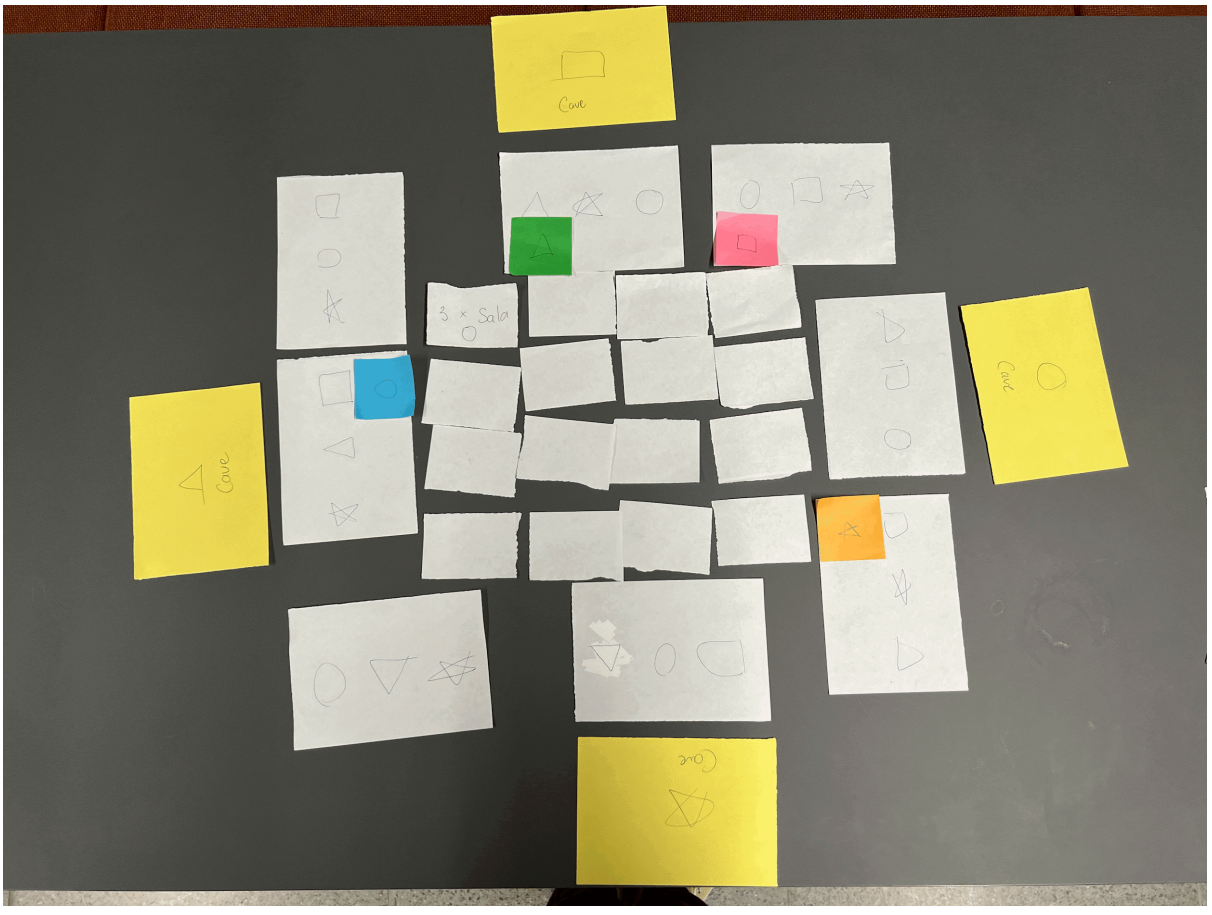


- The chit card then returns to its initial stage, and the turn ends.

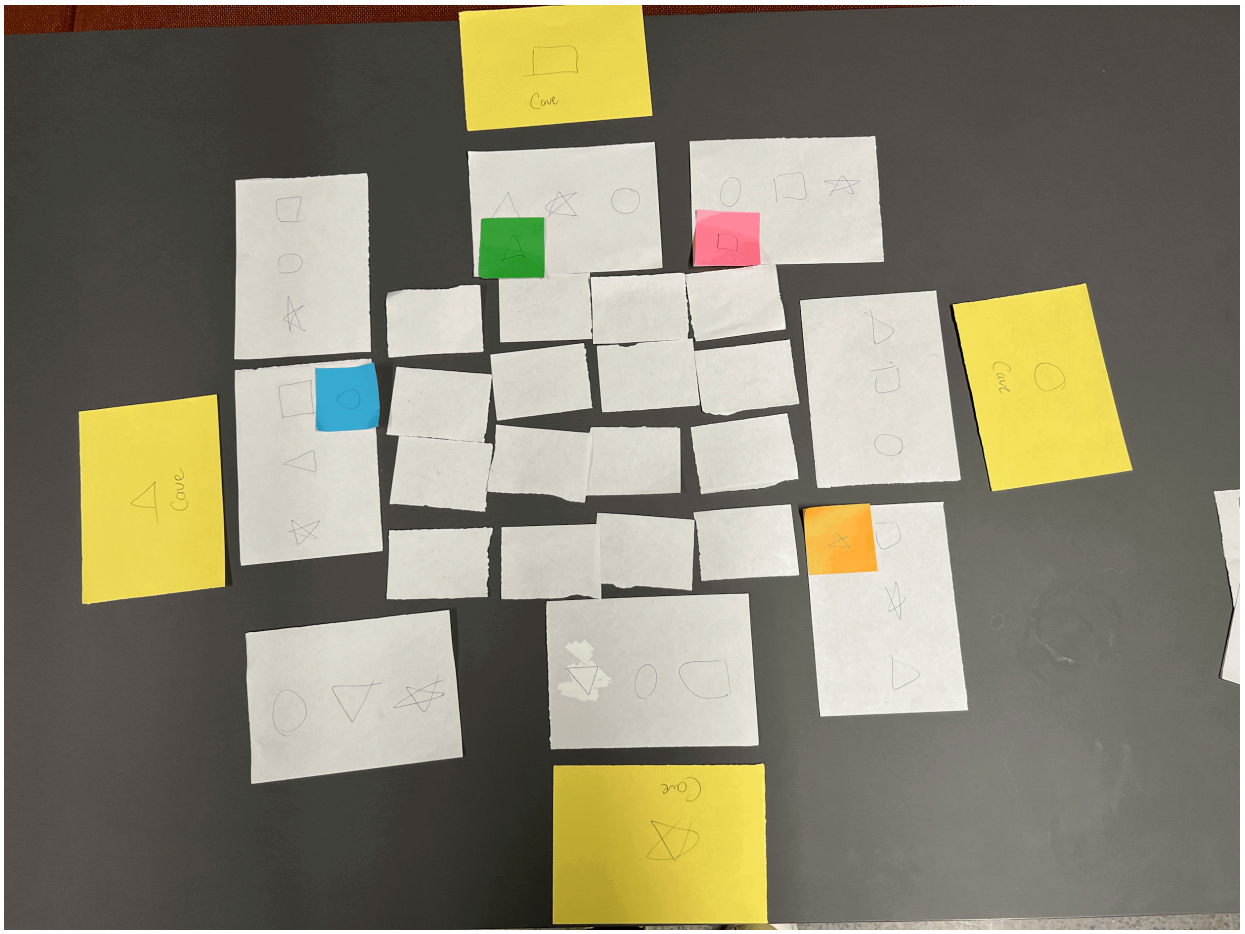


- The circle player picks up the 3-circle chit card which matches the volcano card. It moves straight forward to the square volcano card instead of the triangle cave. This proves that the piece will only move to the corresponding cave.

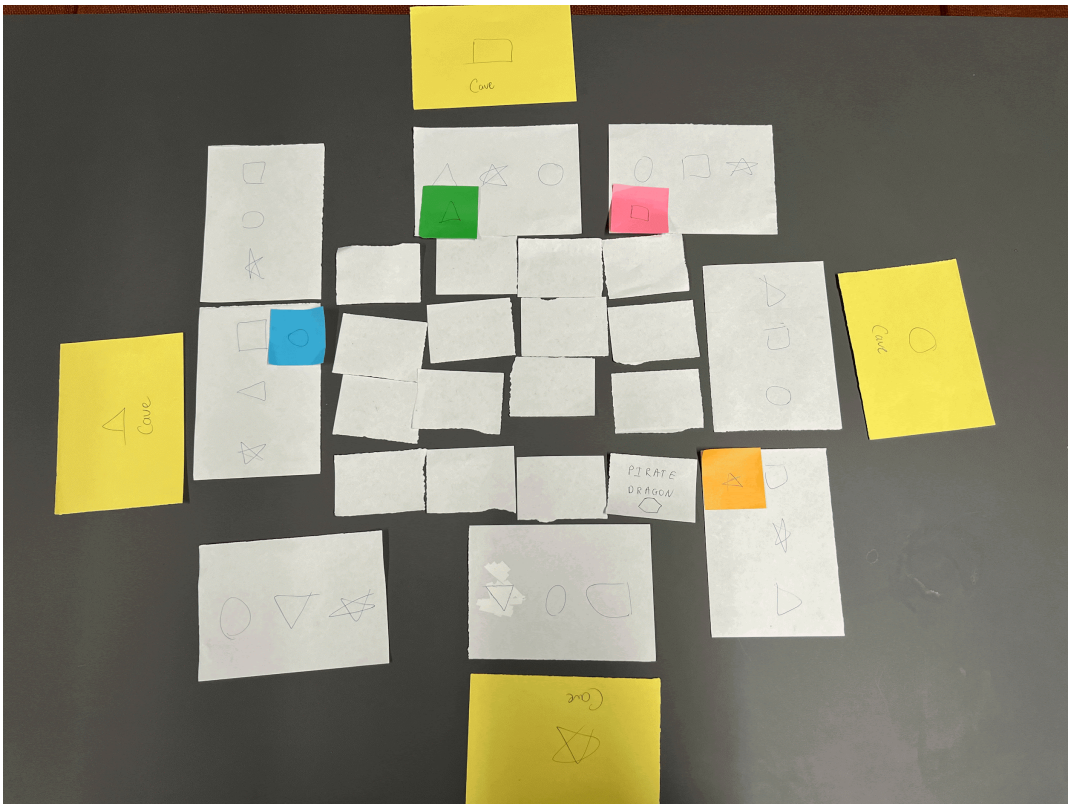


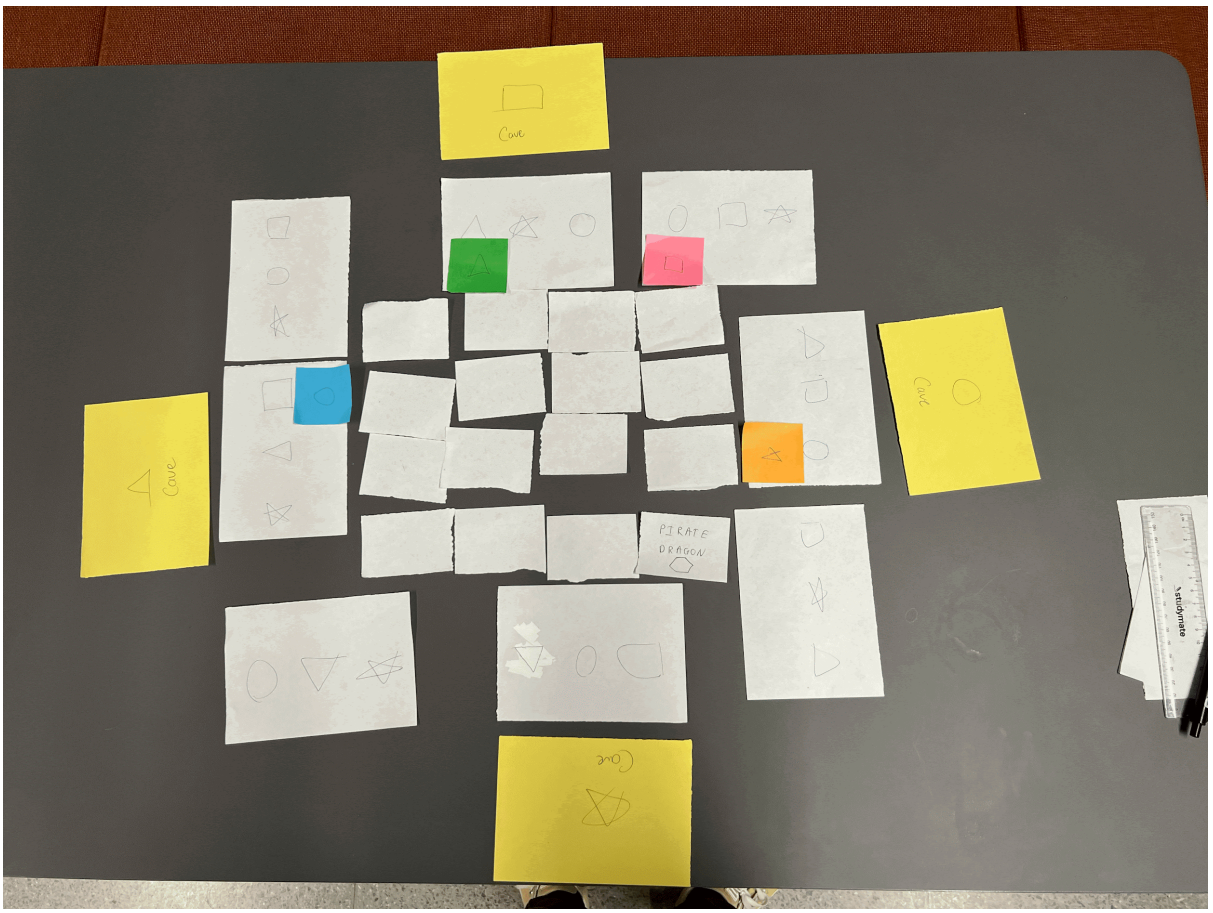


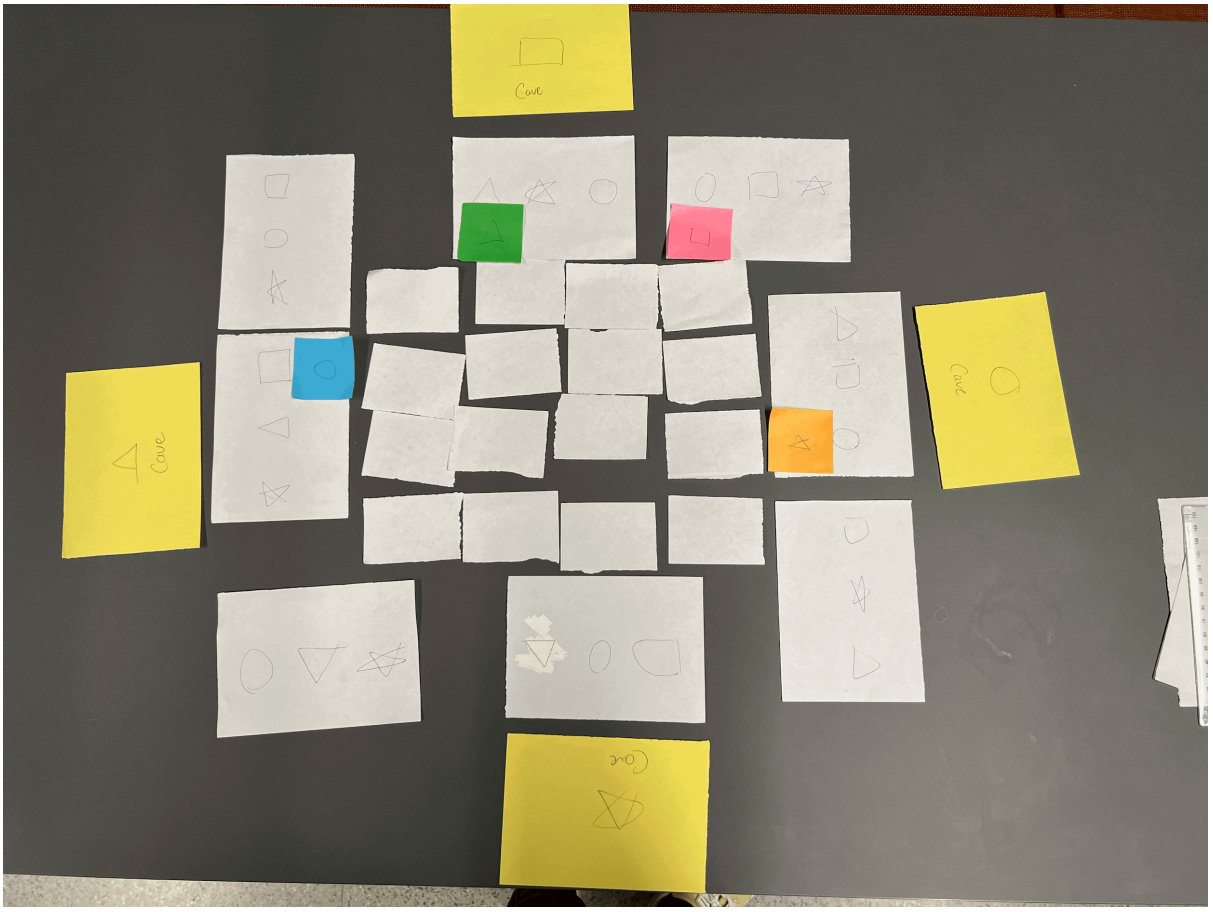
- The chit card then returns to its initial stage, and the turn ends.



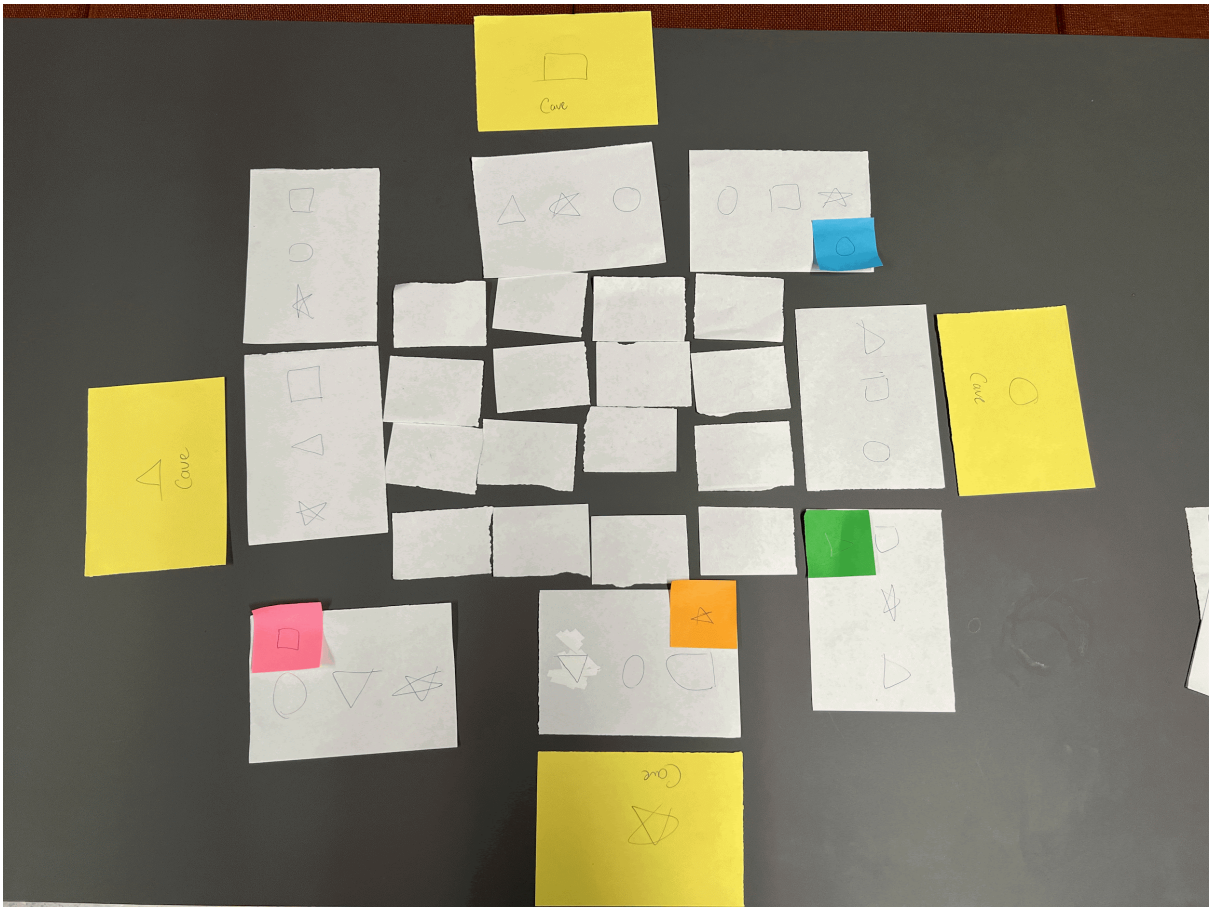
- The triangle player picks up the 1-pirate dragon (1-hexagon) chit card. It moves backward one step to the circle volcano card. The chit card then returns to its initial stage, and the turn ends.

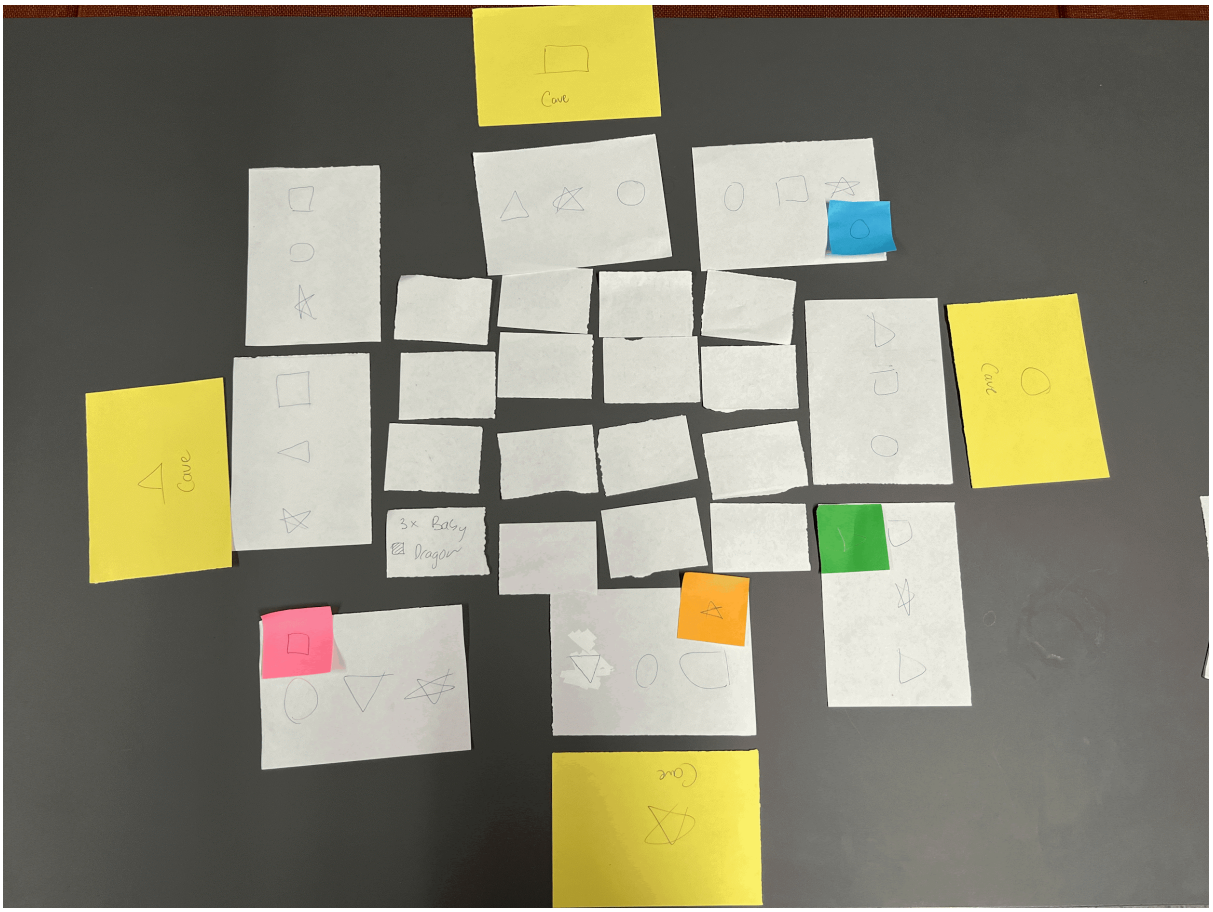


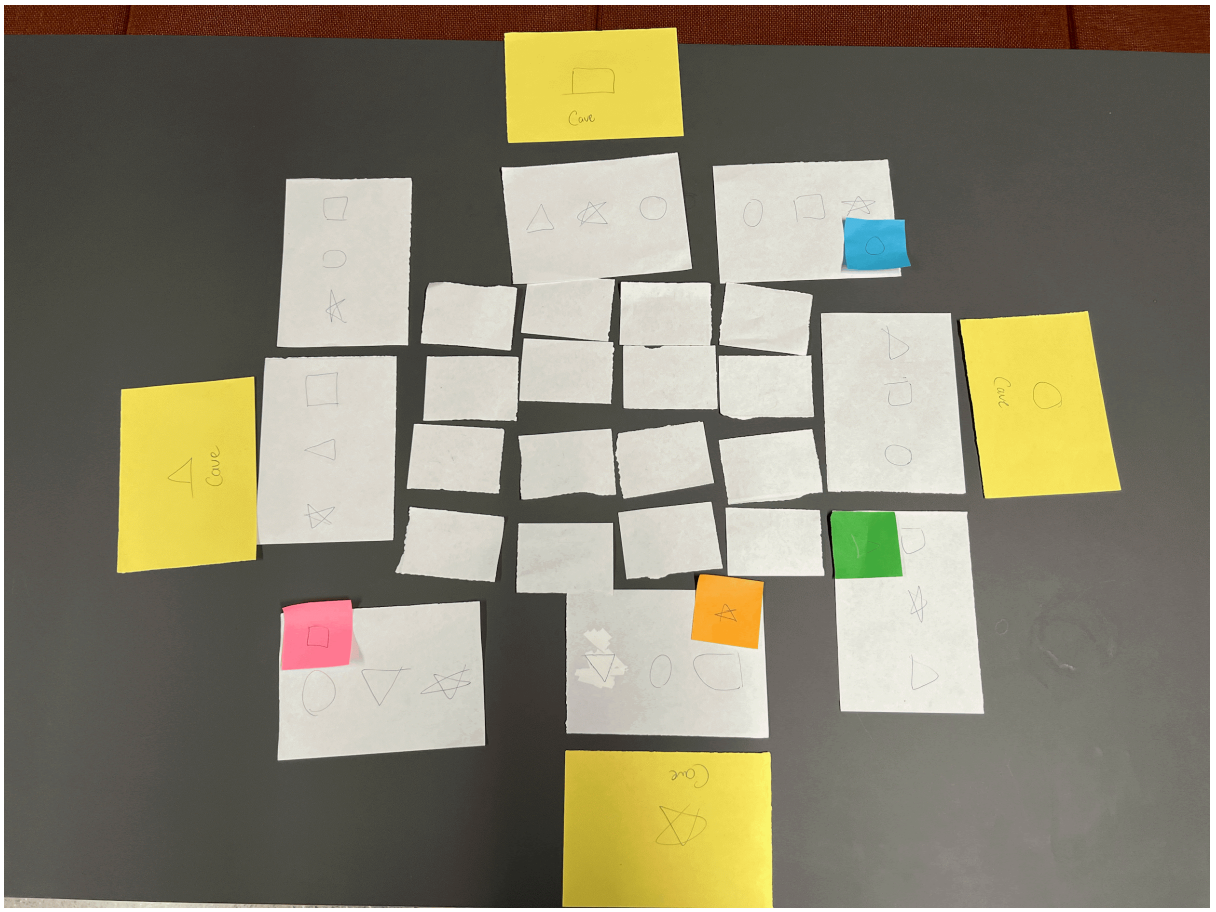

Cave
Cave
Cave
PIRATE
DRAGON
Cave


Cave
Cave
Cave
Cave

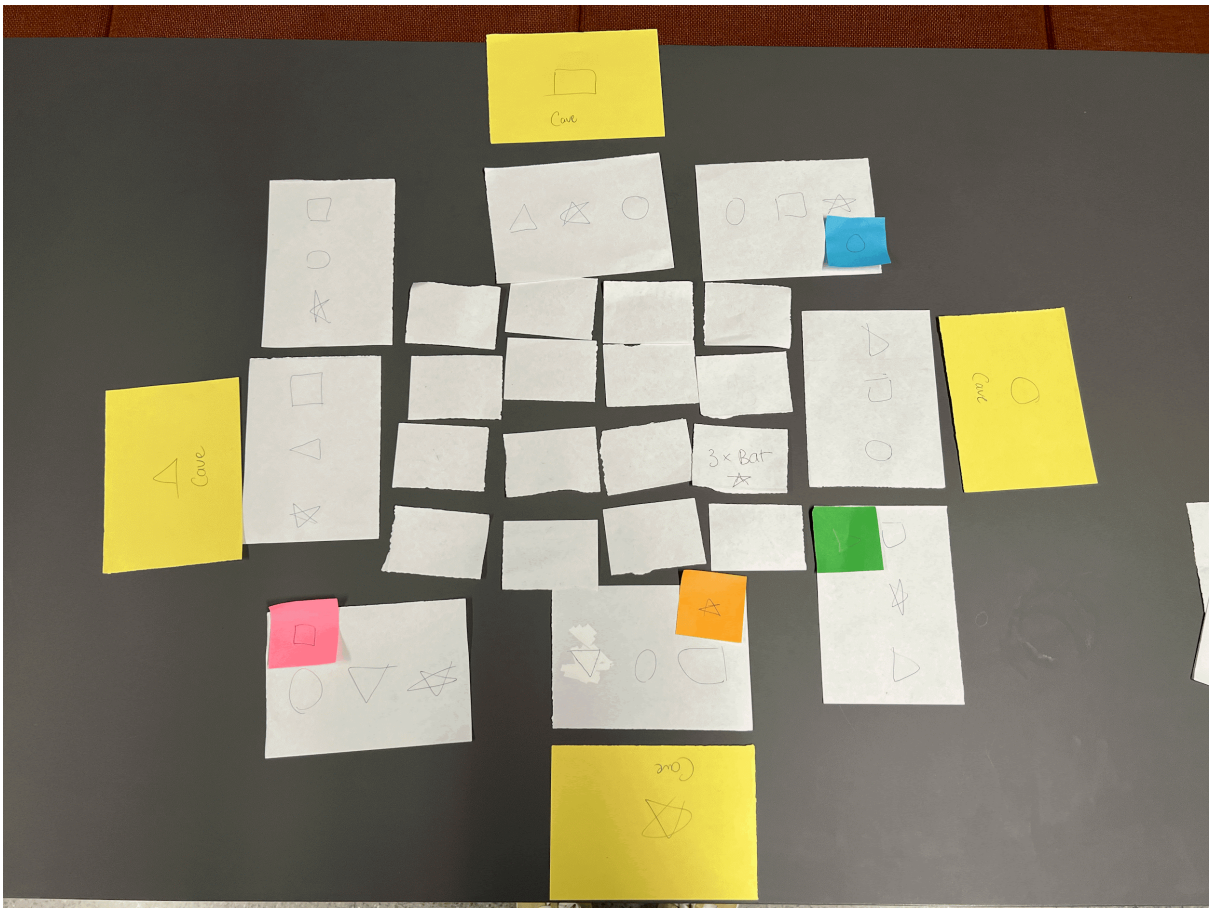
- The star player picks up the 3-square chit card. It moves toward three steps to the star cave. However, the distance between them is less than the number of steps, so the piece can not move and stand outside. The chit card then returns to its initial stage, and the turn ends.



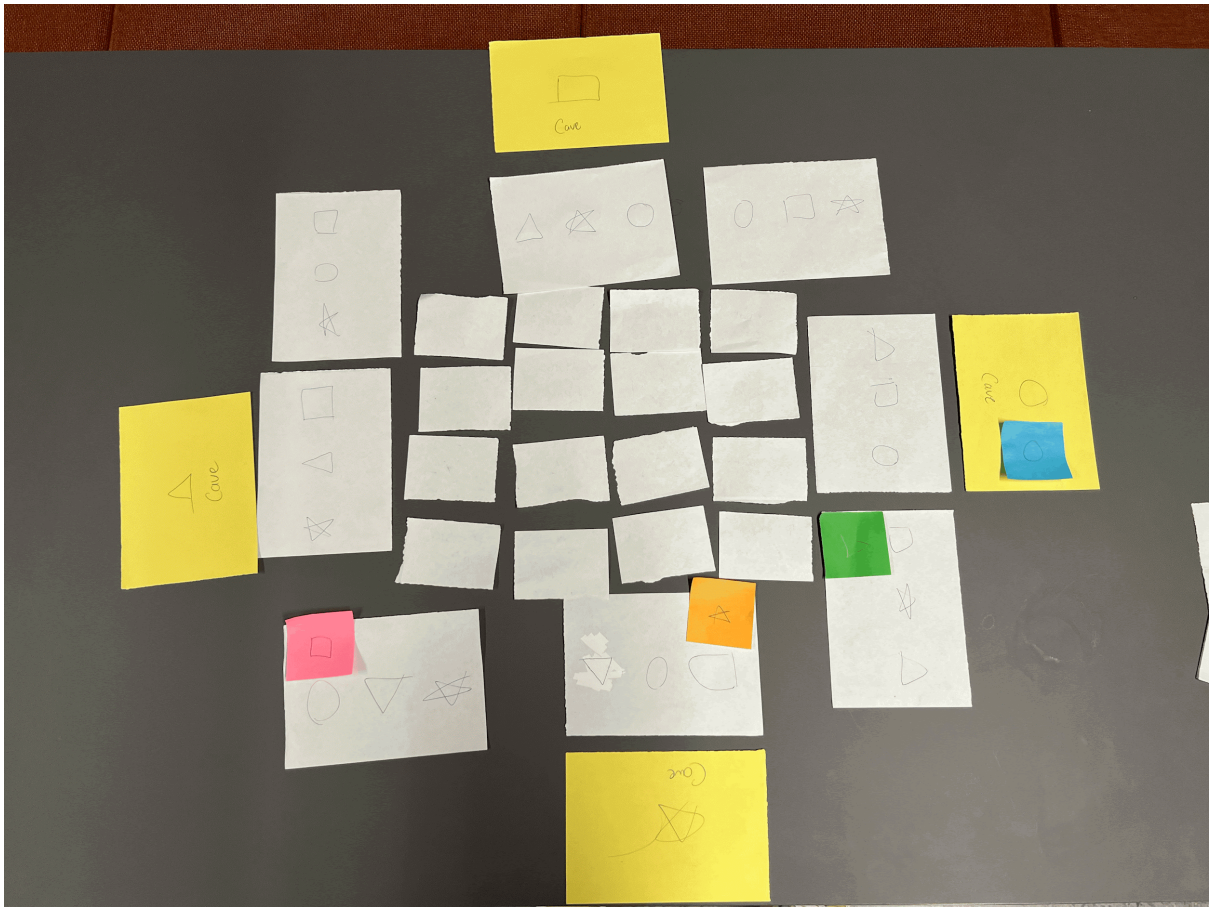




- Finally, the circle player picks up the 3-star chit card which matches the volcano card. It moves straight forward to the circle cave. The chit card returns to its initial stage and the turn ends. The game ends, and the circle wins the game.




Cave
4
Cave
3 × Bat
Cave
Cave


Cave
4
Cave
Cave
Cave