Programming Methodology Project 2 Description

Awesome Yunnori

2024 Spring Semester

Machine Intelligence & Data science LAB

TA Information

- TAs
  - Seokhyeon Jeong (sh102201@snu.ac.kr)
  - Sumin Yu (ysmsoomin@snu.ac.kr)
  - Heewoong Choi (chw0501@snu.ac.kr)

- Post questions about the project on eTL

- DO NOT COPY OR CHEAT (Plagiarism = F grade)

Outline

- C++ grammars for Project2
- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures
- Submission and Grading

Outline

- C++ grammars for Project2

- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures

- Submission and Grading

Machine Intelligence & Data science LAB

C++ grammars for Project2

- STL – pair, vector, map, multiset
- File I/O
- Operator overloading
- Friend functions
- Inheritance
- Virtual functions

Outline

- C++ grammars for Project2
- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures
- Submission and Grading

Machine Intelligence & Data science LAB

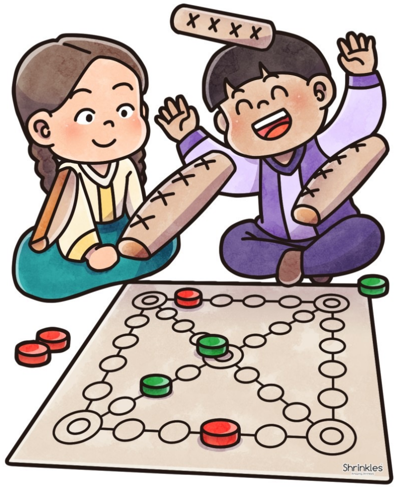
Overview

- Goal: Create a Yunnori game
  - Two types of game

<Basic Yunnori>

2, 3, or 4 normal players

- **w/ 2, 3, or 4 pieces**
- **Typical Yunnori**



<Advanced Yunnori>

2, 3, or 4 Animal players

- **w/ 4 pieces**
- **Each player has special skills**

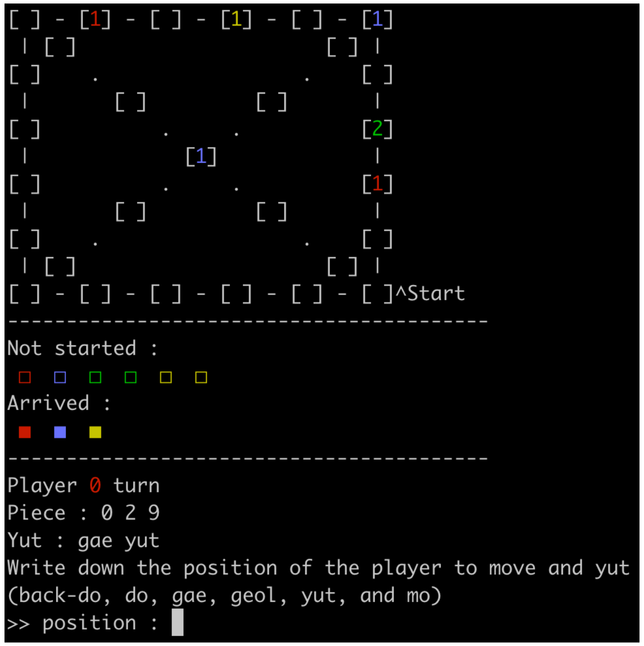
+ Login

+ Pausing, saving, and loading games

Machine Intelligence & Data science LAB

Overview

- Example game screen



```
[ ] - [1] - [ ] - [1] - [ ] - [1]
 | [ ]                   [ ] |
[ ]    .              .    [ ]
 |      [ ]         [ ]      |
[ ]          .    .        [2]
 |             [1]           |
[ ]          .    .        [1]
 |      [ ]         [ ]      |
[ ]    .              .    [ ]
 | [ ]                   [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
----------------------------------------
Not started :
 □  □  □  □  □  □
Arrived :
 ■  ■  ■
----------------------------------------
Player 0 turn
Piece : 0 2 9
Yut : gae yut
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position :
```

- You will practice:
  - utilizing inheritance, virtual functions, friend functions, etc…
  - file I/O, terminal I/O
  - managing the flow of a big program
  - debugging for a perfect implementation

Machine Intelligence & Data science LAB

Outline

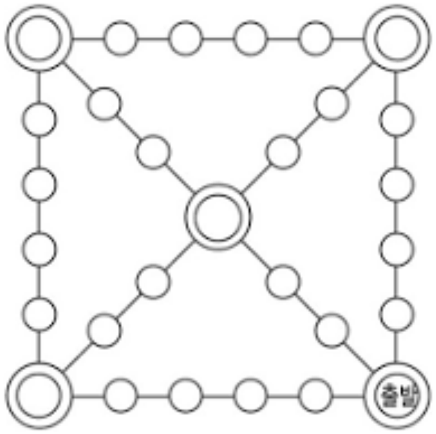
- C++ grammars for Project2
- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures
- Submission and Grading

Introduction to the gameplay

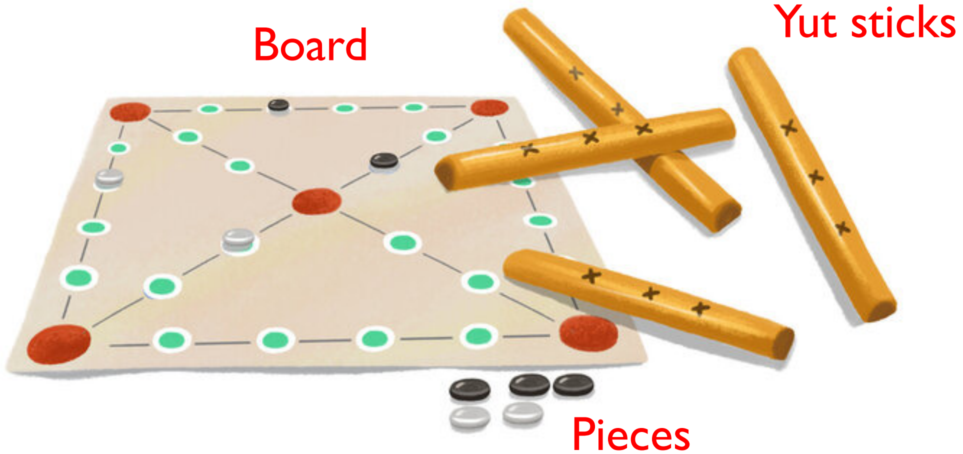
- Game components
- Game-end condition
- Rules of throwing Yut sticks
- Gameplay mechanics

Introduction to the gameplay

- Game components
  - Game board
  - Game pieces for each player
  - Yut sticks
    - Pieces are moved based on the results from throwing 4 Yut sticks.

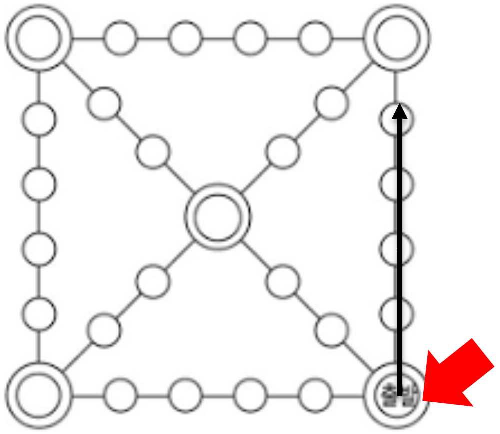


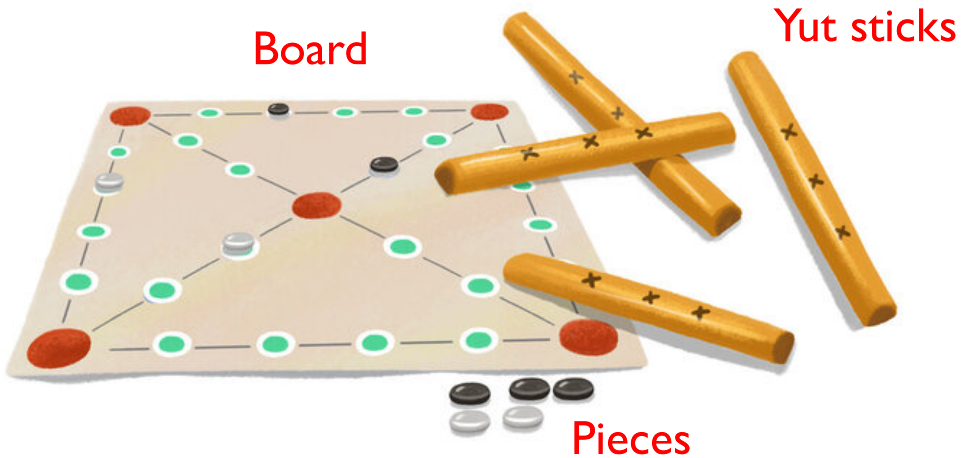
Board



Introduction to the gameplay

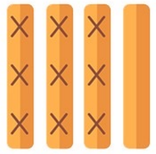
- Game-end condition
  - Piece arrival = a piece has traveled around the board and **passed through** the ending point
  - If all the player's pieces arrive, that player wins the game and the game ends.

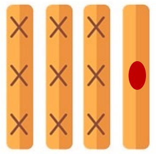

Board



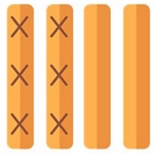
Introduction to the gameplay

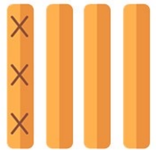
- Rules of throwing Yut sticks

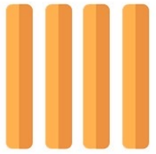

do: move one space.

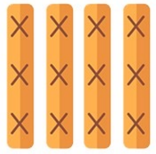

back-do: move one space back.

Only one of the four yut sticks is marked.


gae : move two spaces.


geol : move three spaces.

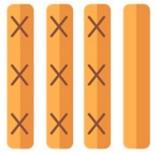

yut : move four spaces and throw again.


mo : move five spaces and throw again.

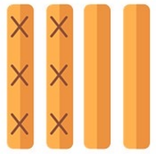
* Assume that the stick is equally likely to come up front or back.

Introduction to the gameplay

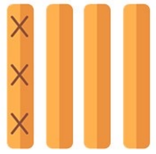
- For clarity, the probabilities of each yut result are



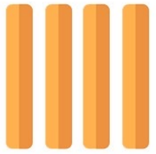
do: $\frac{3}{16}$



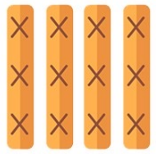
gae : $\frac{3}{8}$



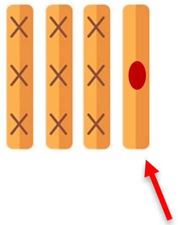
geol : $\frac{1}{4}$



yut : $\frac{1}{16}$



mo : $\frac{1}{16}$



back-do: $\frac{1}{16}$

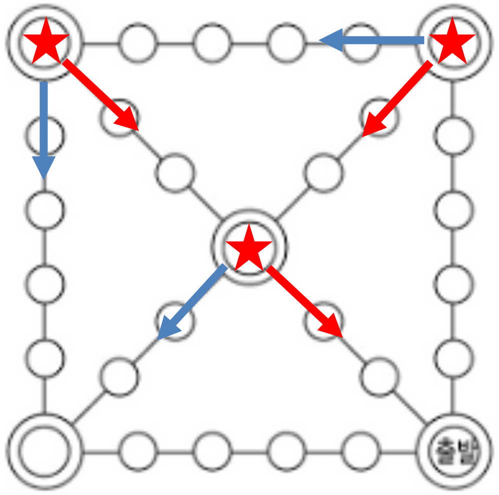
Only one of the four yut sticks is marked.

* Assume that the stick is equally likely to come up front or back.

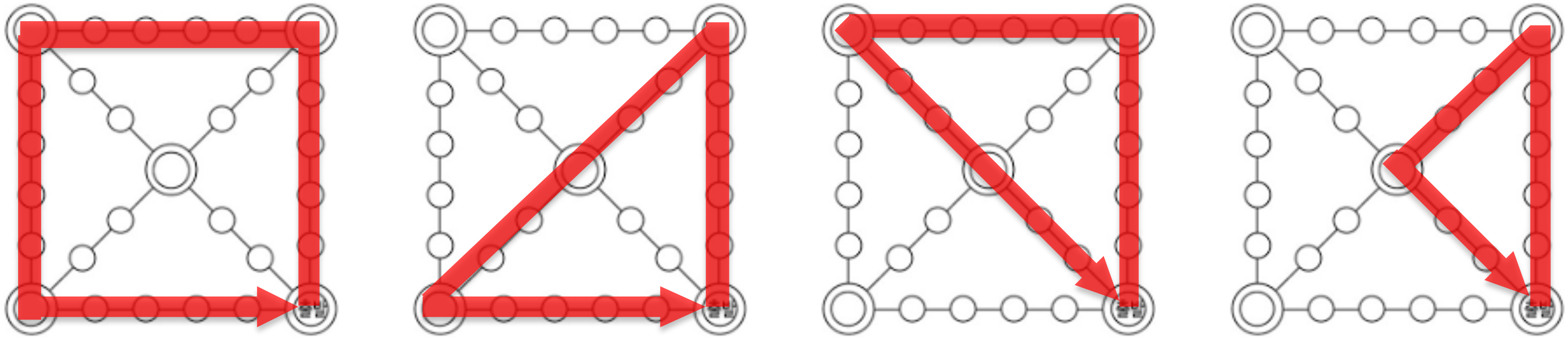
Machine Intelligence & Data science LAB

Introduction to the gameplay

- Gameplay mechanics
  - Movement
    - If the piece stops exactly at ★ ,
      it must move in the direction of → .
      Otherwise, it must move in the direction of → .



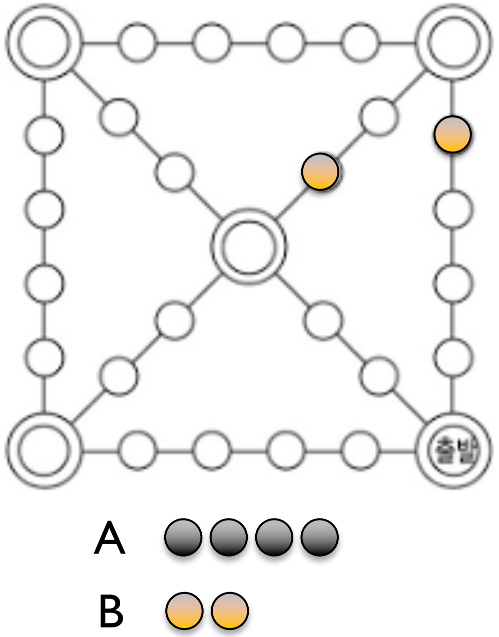
    - There are 4 possible directions to move. (without considering back-do)



Machine Intelligence & Data science LAB

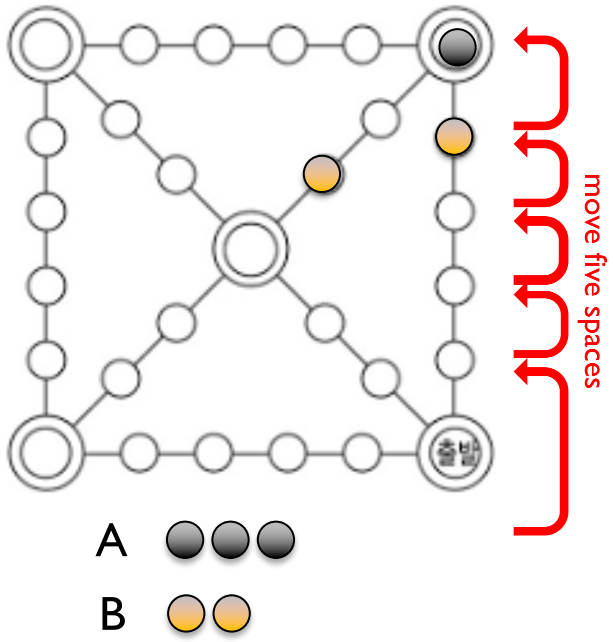
Introduction to the gameplay

- For example,
  - It's player A's turn, and the pieces haven't departed yet.



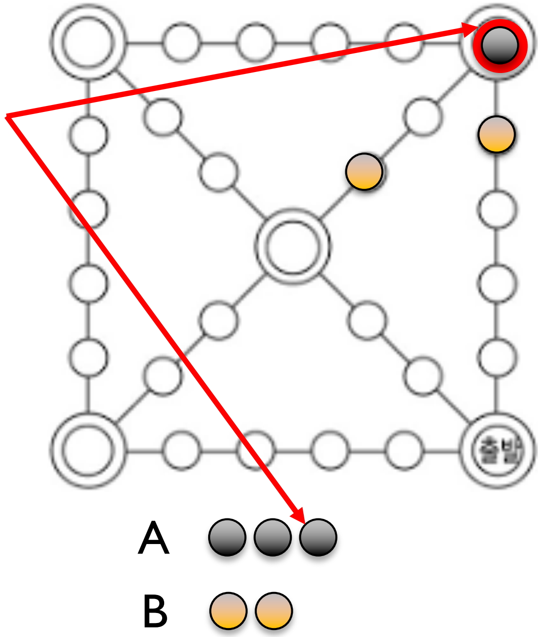
Introduction to the gameplay

- For example,
  - It’s player A's turn, and the pieces haven’t departed yet.
  - Player A throws the yut sticks and the result is mo.



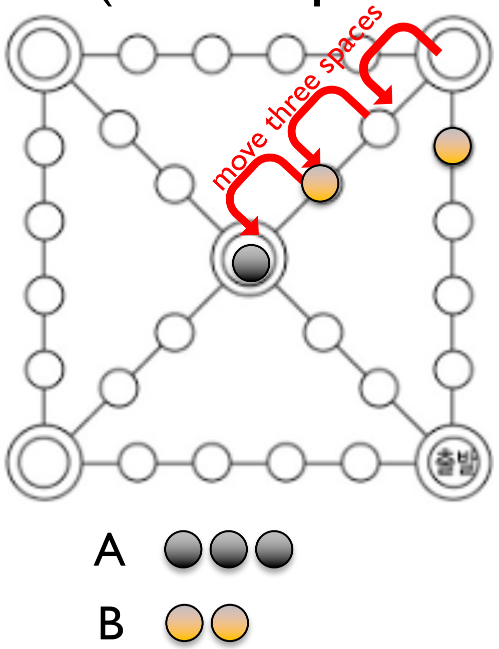
Introduction to the gameplay

- For example,
  - It's player A's turn, and the pieces haven't departed yet.
  - Player A throws the yut sticks and the result is mo.
  - Player A throws the yut sticks again (b/c the previous result is mo) and the result is geol.
    - Player A choose a piece to move



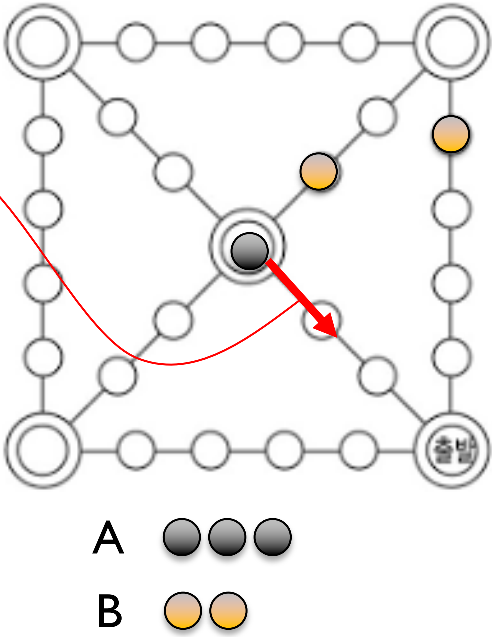
Introduction to the gameplay

- For example,
  - It's player A's turn, and the pieces haven't departed yet.
  - Player A throws the yut sticks and the result is mo.
  - Player A throws the yut sticks again (b/c the previous result is mo) and the result is geol.
    - Player A choose a piece to move



Introduction to the gameplay

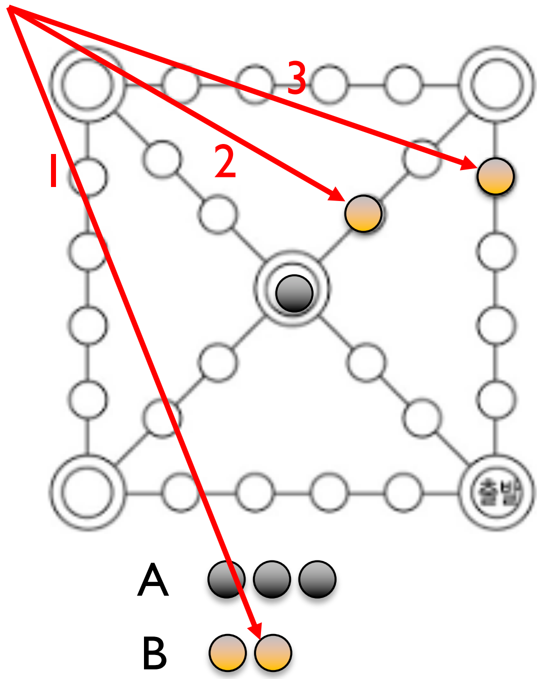
- For example,
  - It's player A's turn, and the pieces haven't departed yet.
  - Player A throws the yut sticks and the result is mo.
  - Player A throws the yut sticks again (b/c the previous result is mo) and the result is geol.
  - Player A's turn is over.
    - player A must move in this direction on his next turn.



Machine Intelligence & Data science LAB

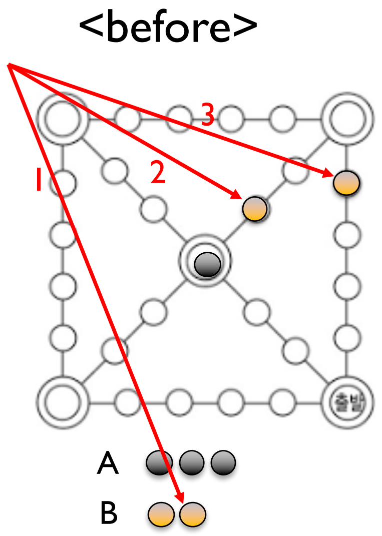
Introduction to the gameplay

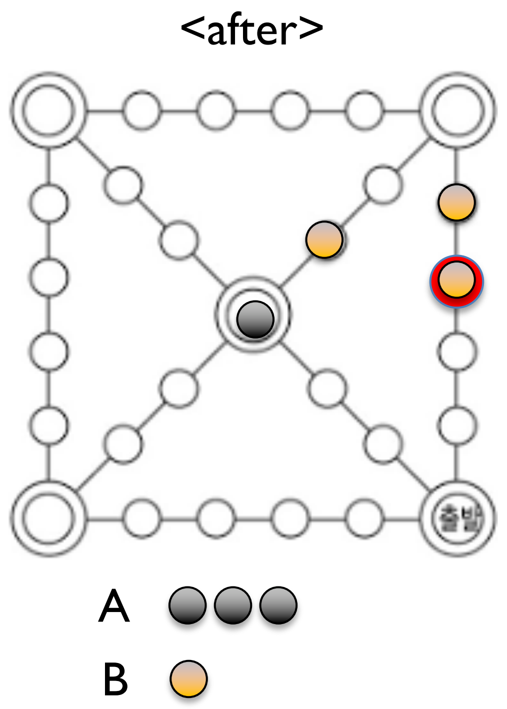
- For example,
  - It's player B's turn, and the two pieces haven't departed yet.
  - Player B throws the yut sticks and the result is geol.
    - Player B choose a piece to move



Introduction to the gameplay

- For example,
  - It's player B's turn, and the two pieces haven't departed yet.
  - Player B throws the yut sticks and the result is geol.
    - Player B choose a piece to move
      - If B choose piece 1

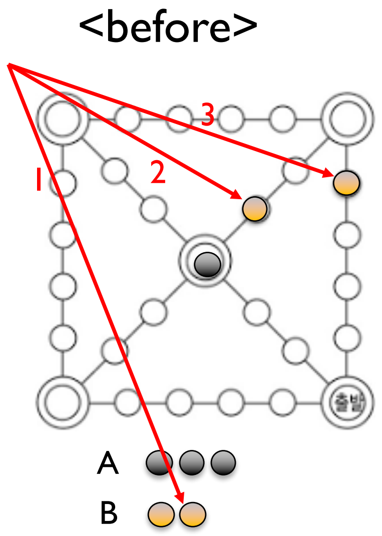


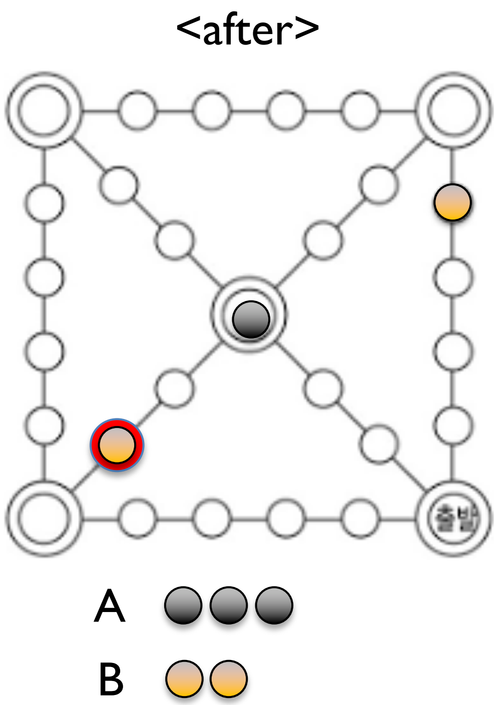


Machine Intelligence & Data science LAB

Introduction to the gameplay

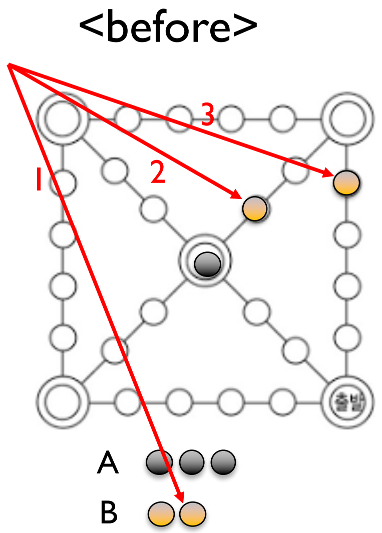
- For example,
  - It's player B's turn, and the two pieces haven't departed yet.
  - Player B throws the yut sticks and the result is geol.
    - Player B choose a piece to move
      - If B choose piece 2

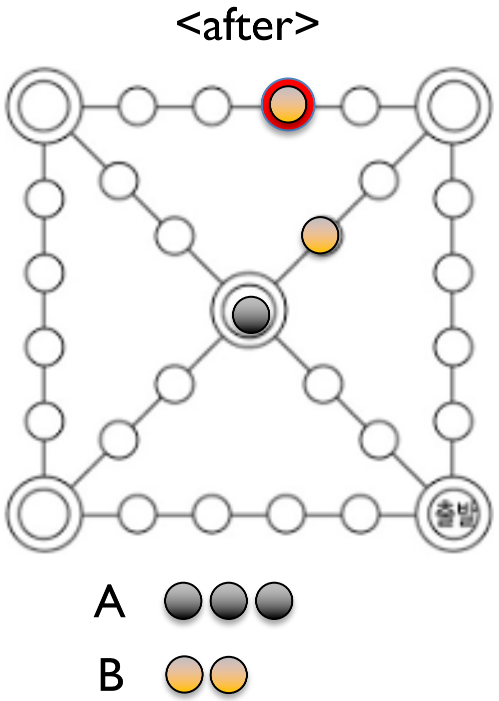




Introduction to the gameplay

- For example,
  - It's player B's turn, and the two pieces haven't departed yet.
  - Player B throws the yut sticks and the result is geol.
    - Player B choose a piece to move
      - If B choose piece 3





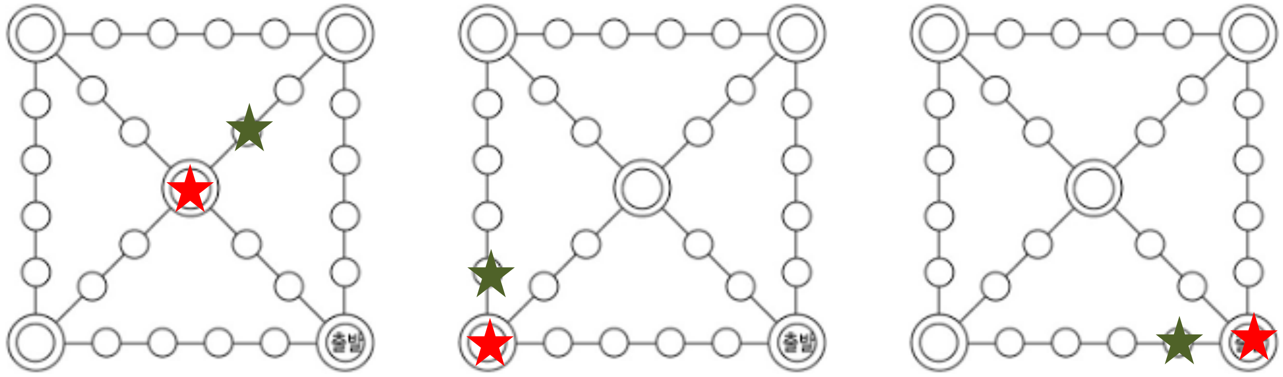
Introduction to the gameplay

- Gameplay mechanics
  - Movement
    - Back-do: A piece on the board that you choose to move must go back one space.

✓ You cannot put a new token on the board with Back-do.

✓ Things that may be confusing:

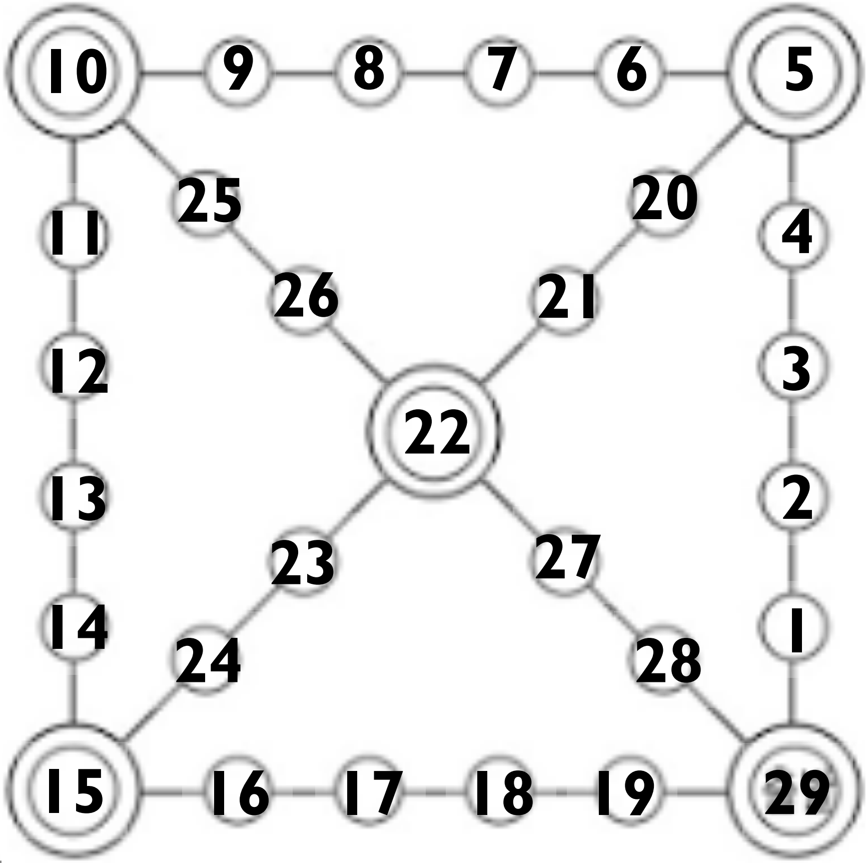
- If Back-do occurs when a piece is on ★, move it to ★ .



Introduction to the gameplay

❖ These numbers are important because they are directly related to our grading.

- We specify each position of the board.



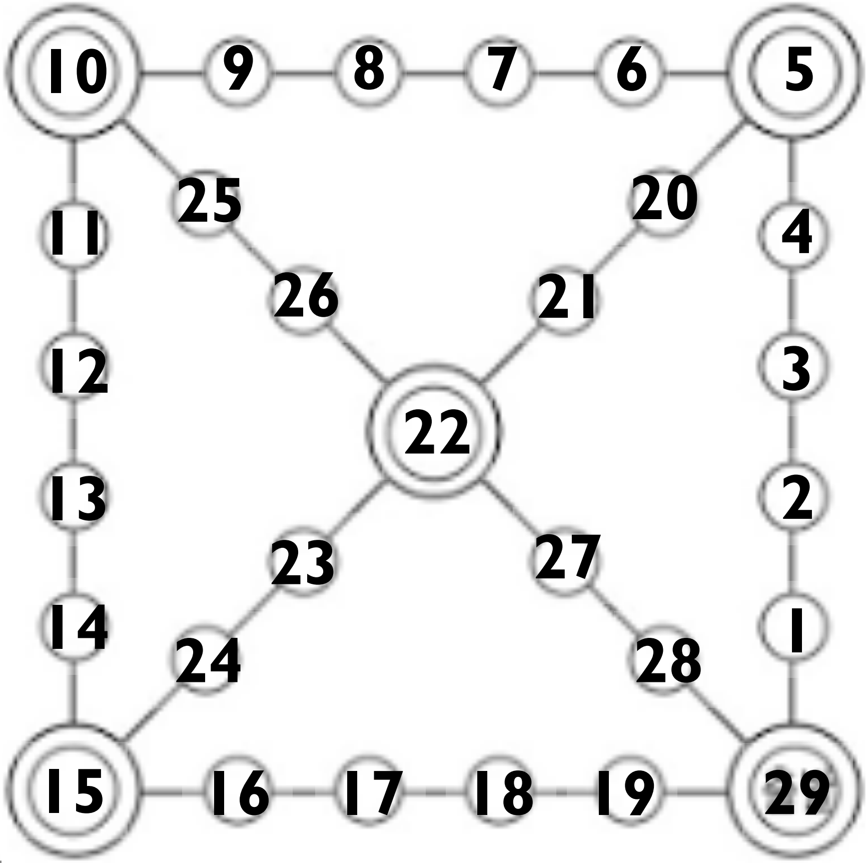
Position 0
: not yet departed

Position 29
: The piece needs to go at least one more space from 29
to be considered arrived.

Introduction to the gameplay

This page includes the content on p25.

- To clarify back-do movement



Position 0 → cannot move
Position 1 → 29
Position 29 → 19
Position 15 → 14
Position 22 → 21

Otherwise, for example,
Position 10 → 9
Position 19 → 18
Position 26 → 25
Position 23 → 22
…

Machine Intelligence & Data science LAB

Introduction to the gameplay

- Gameplay mechanics
  - "Catching"

    when a player lands on a space occupied by a piece of one of his opponents.

    [results]

    - The opponent's piece is eliminated from the board (position becomes 0) and it should start over.
    - The player throws again.

  - "Stacking"

    when a player lands on a space occupied by his pieces.

    [results]

    - The player should stack their pieces to move together as one unit.

Machine Intelligence & Data science LAB

Introduction to the gameplay

- Gameplay mechanics
  - "Throwing again"

    when a player gets yut or mo.

[results]

- The player throws again and moves his pieces with the stick results.

  *It doesn't matter what order the stick results are applied.

For example,

player A's turn:

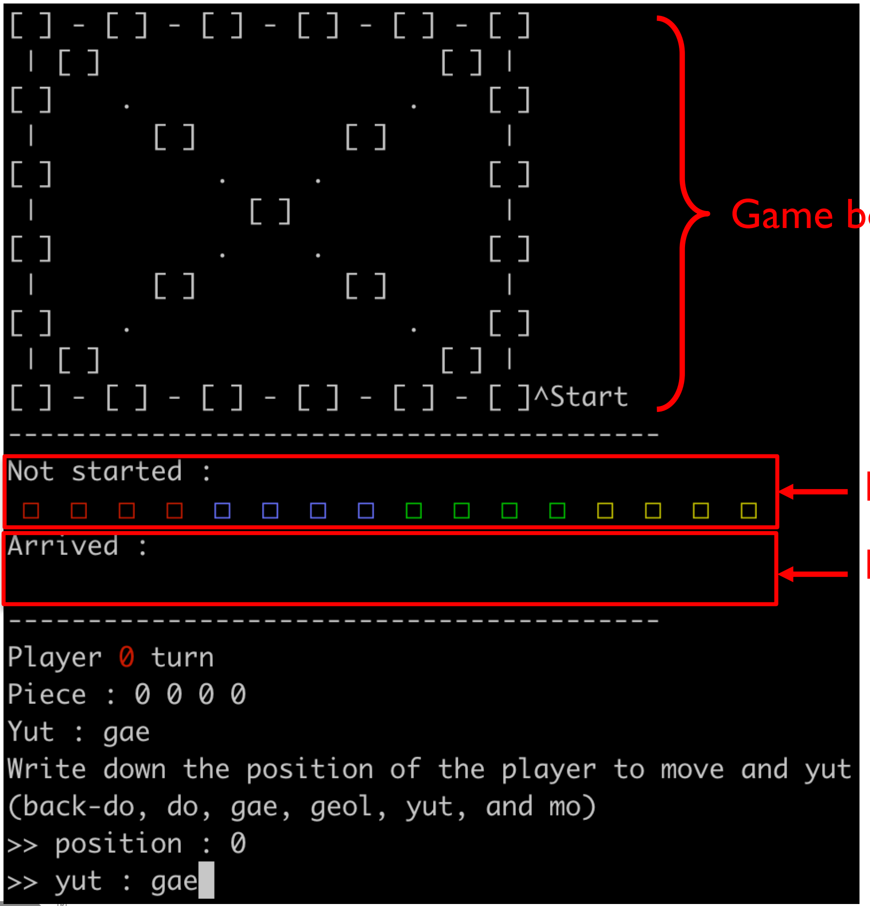
1st throwing result is Yut → 2nd throwing result is Mo → 3rd throwing result is Goel

➡ player A chooses and moves his tokens based on the stick results (Yut, Mo, and Geol).

The order of application of Yut, Mo, and Goel doesn't matter.

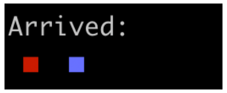
Gameplay examples

- Game visualization



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |       [ ]         [ ]       |
[ ]           .   .           [ ]
 |             [ ]             |
[ ]           .   .           [ ]
 |       [ ]         [ ]       |
[ ]    .                 .    [ ]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 0 turn
Piece : 0 0 0 0
Yut : gae
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : gae
```

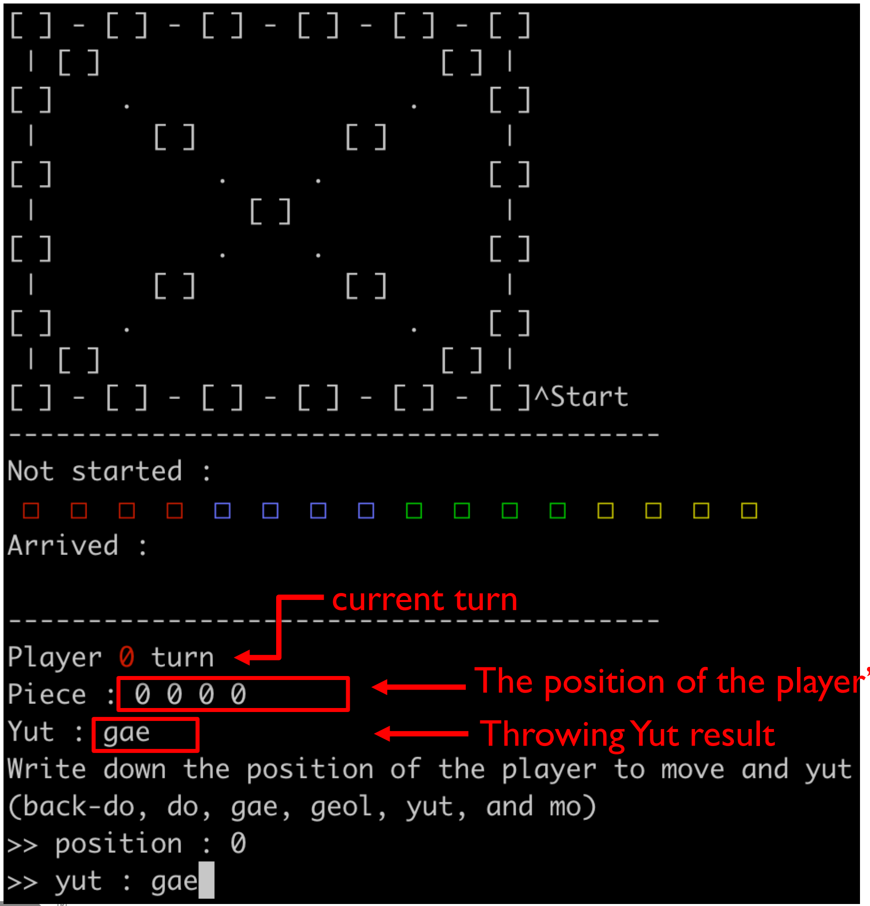
Game board
Players' pieces that have not yet departed
Players' pieces that have arrived



```
Arrived:
 🟥  🟦
```

Gameplay examples

- Game visualization

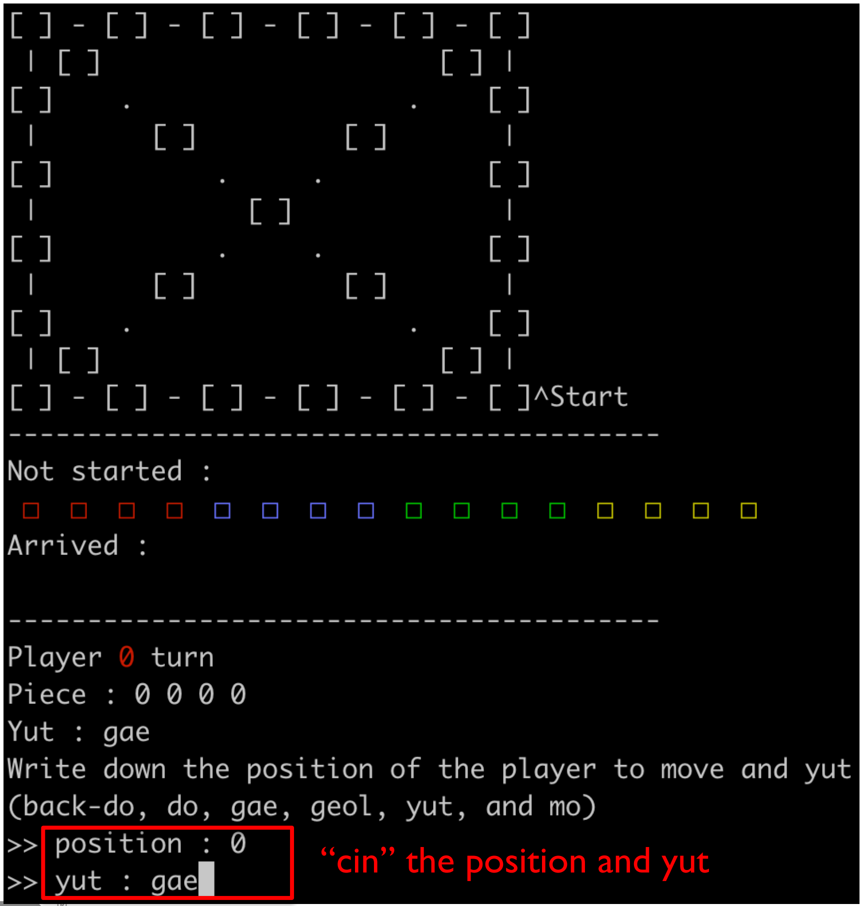


```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |      [ ]         [ ]        |
[ ]          .    .           [ ]
 |             [ ]             |
[ ]          .    .           [ ]
 |      [ ]         [ ]        |
[ ]    .                 .    [ ]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 0 turn
Piece : 0 0 0 0
Yut : gae
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : gae
```

current turn
The position of the player's pieces on the current turn (Exclude arrived pieces)
Throwing Yut result

Gameplay examples

- Game visualization

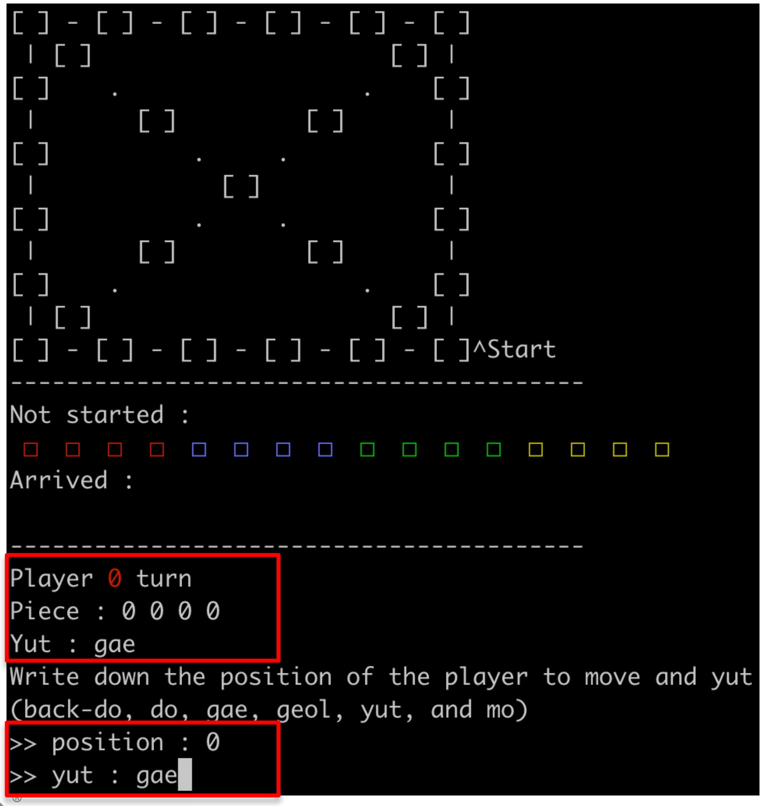


```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                   [ ] |
[ ]    .               .    [ ]
 |      [ ]         [ ]      |
[ ]          .     .         [ ]
 |             [ ]           |
[ ]          .     .         [ ]
 |      [ ]         [ ]      |
[ ]    .               .    [ ]
 | [ ]                   [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 0 turn
Piece : 0 0 0 0
Yut : gae
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : gae
```

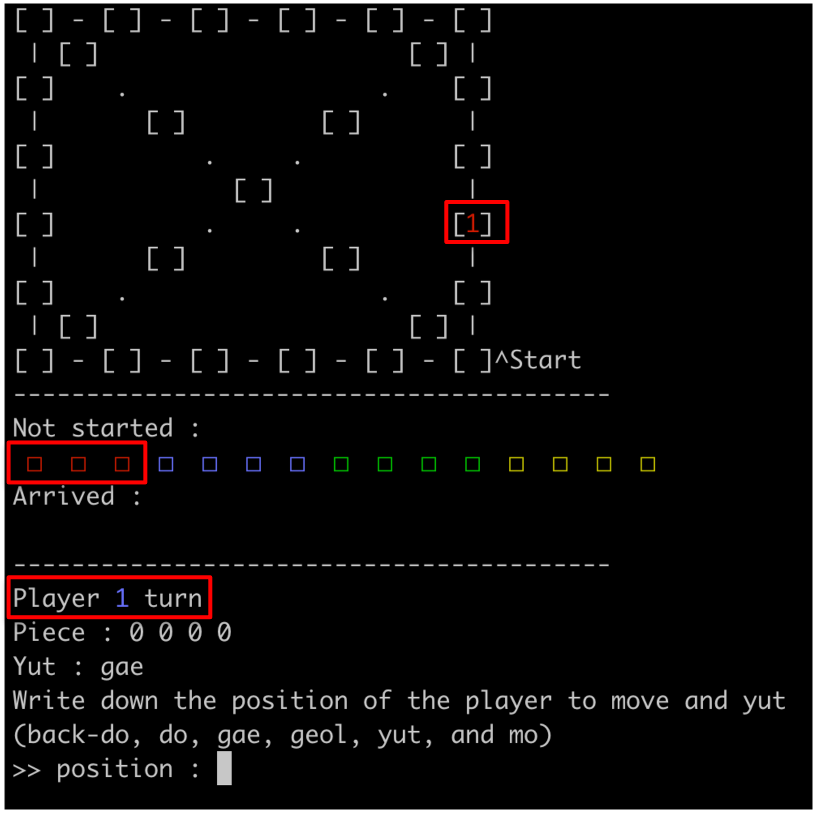
"cin" the position and yut

Gameplay examples

- Game visualization (ex 1)
  - player 0 moves with gae



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                   [ ] |
[ ]    .              .    [ ]
 |      [ ]         [ ]      |
[ ]           .    .       [ ]
 |              [ ]          |
[ ]           .    .       [ ]
 |      [ ]         [ ]      |
[ ]    .              .    [ ]
 | [ ]                   [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
Arrived :

-----------------------------------------
Player 0 turn
Piece : 0 0 0 0
Yut : gae
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : gae
```

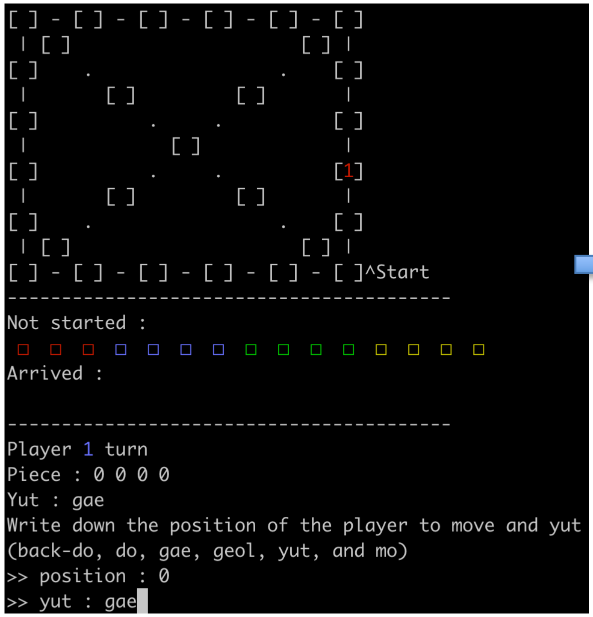


```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                   [ ] |
[ ]    .              .    [ ]
 |      [ ]         [ ]      |
[ ]           .    .       [ ]
 |              [ ]          |
[ ]           .    .       [1]
 |      [ ]         [ ]      |
[ ]    .              .    [ ]
 | [ ]                   [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □ □ □ □ □ □ □ □ □ □ □ □ □ □ □
Arrived :

-----------------------------------------
Player 1 turn
Piece : 0 0 0 0
Yut : gae
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position :
```

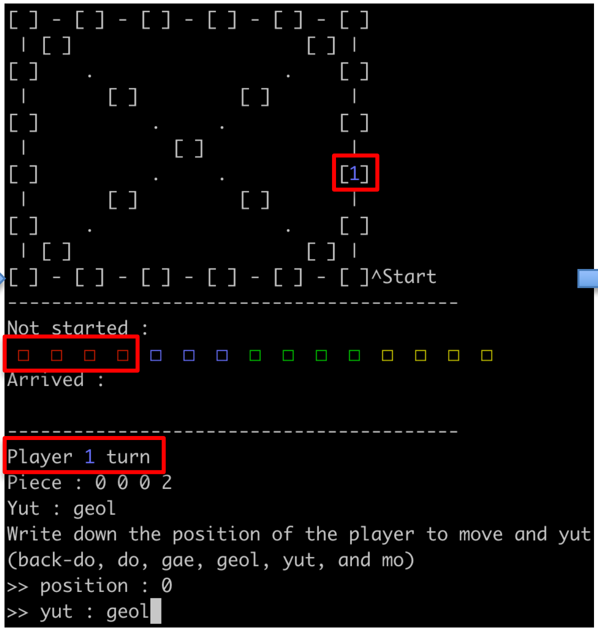
Machine Intelligence & Data science LAB

Gameplay examples

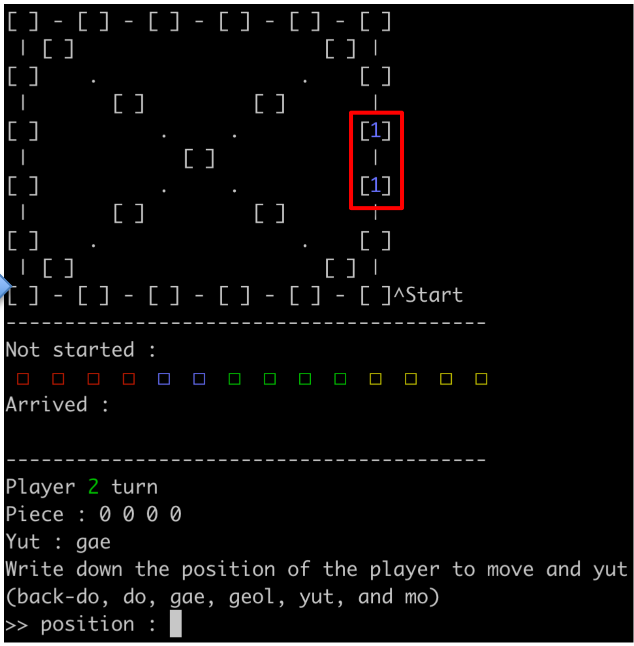
- Game visualization (ex 2)
  - player 1 moves with gae → catches player 0
    → throws yut sticks again → moves with geol



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |       [ ]         [ ]       |
[ ]          .     .          [ ]
 |             [ ]             |
[ ]          .     .          [1]
 |       [ ]         [ ]       |
[ ]    .                 .    [ ]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 1 turn
Piece : 0 0 0 0
Yut : gae
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : gae
```



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |       [ ]         [ ]       |
[ ]          .     .          [ ]
 |             [ ]             |
[ ]          .     .          [1]
 |       [ ]         [ ]       |
[ ]    .                 .    [ ]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 1 turn
Piece : 0 0 0 2
Yut : geol
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : geol
```



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |       [ ]         [ ]       |
[ ]          .     .          [1]
 |             [ ]             |
[ ]          .     .          [1]
 |       [ ]         [ ]       |
[ ]    .                 .    [ ]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 2 turn
Piece : 0 0 0 0
Yut : gae
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position :
```

Gameplay examples

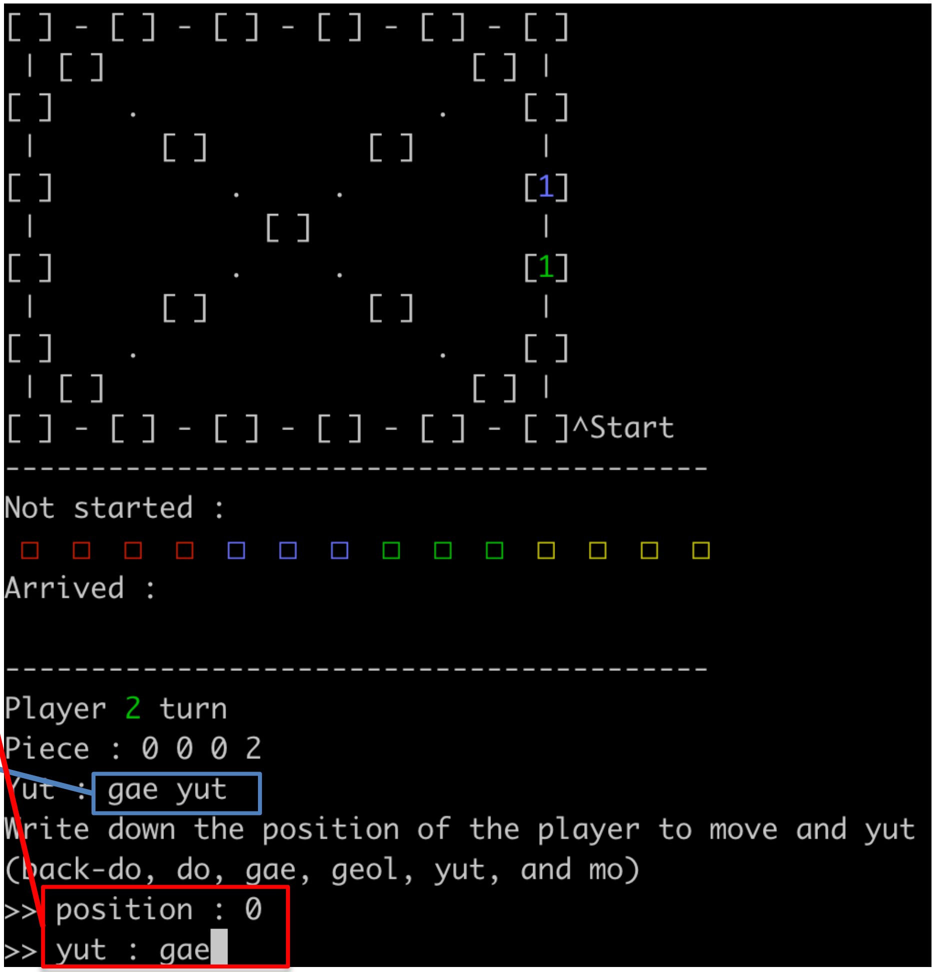
- Game visualization (ex 3)
  - player 2 throws yut sticks and the result is yut

  → throws again and the result is gae

  → moves a new piece with gae (stacking happens)

❖ Throw until no yut or mo comes up, and print a list of the yut results at once

❖ Printing rule: Print in the order of back-do, do, gae, geol, yut, mo



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |      [ ]         [ ]        |
[ ]           .   .           [1]
 |             [ ]             |
[ ]           .   .           [1]
 |      [ ]         [ ]        |
[ ]    .                 .    [ ]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 2 turn
Piece : 0 0 0 2
Yut : gae yut
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : gae
```

Gameplay examples

- Game visualization (ex 3)
  - player 2 throws yut sticks and the result is yut

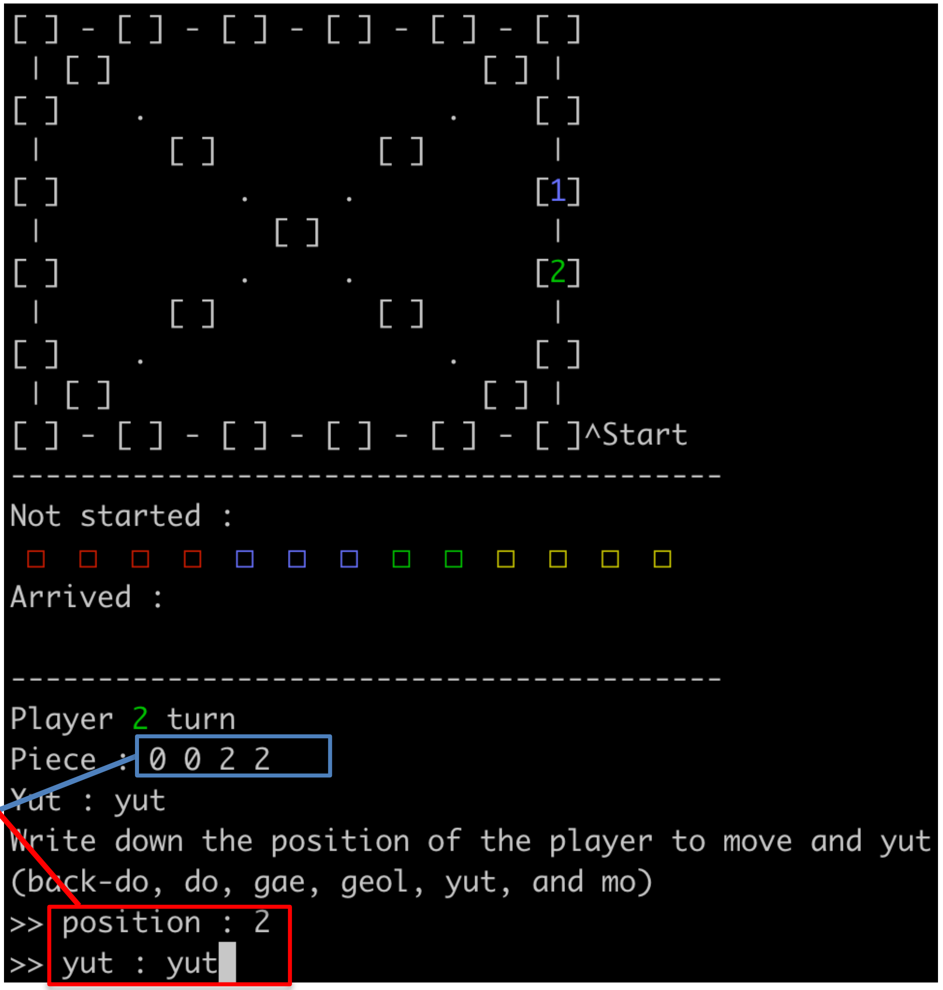
→ throws again and the result is gae

→ moves a new piece with gae (stacking happens)

→moves pieces at 2 with yut

❖ Piece: 0 0 2 2

(two pieces that haven't departed yet, two pieces at position 2)



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                   [ ] |
[ ]     .                 .    [ ]
 |        [ ]         [ ]        |
[ ]           .     .          [1]
 |              [ ]              |
[ ]           .     .          [2]
 |        [ ]         [ ]        |
[ ]     .                 .    [ ]
 | [ ]                   [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 2 turn
Piece : 0 0 2 2
Yut : yut
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 2
>> yut : yut
```

Gameplay examples

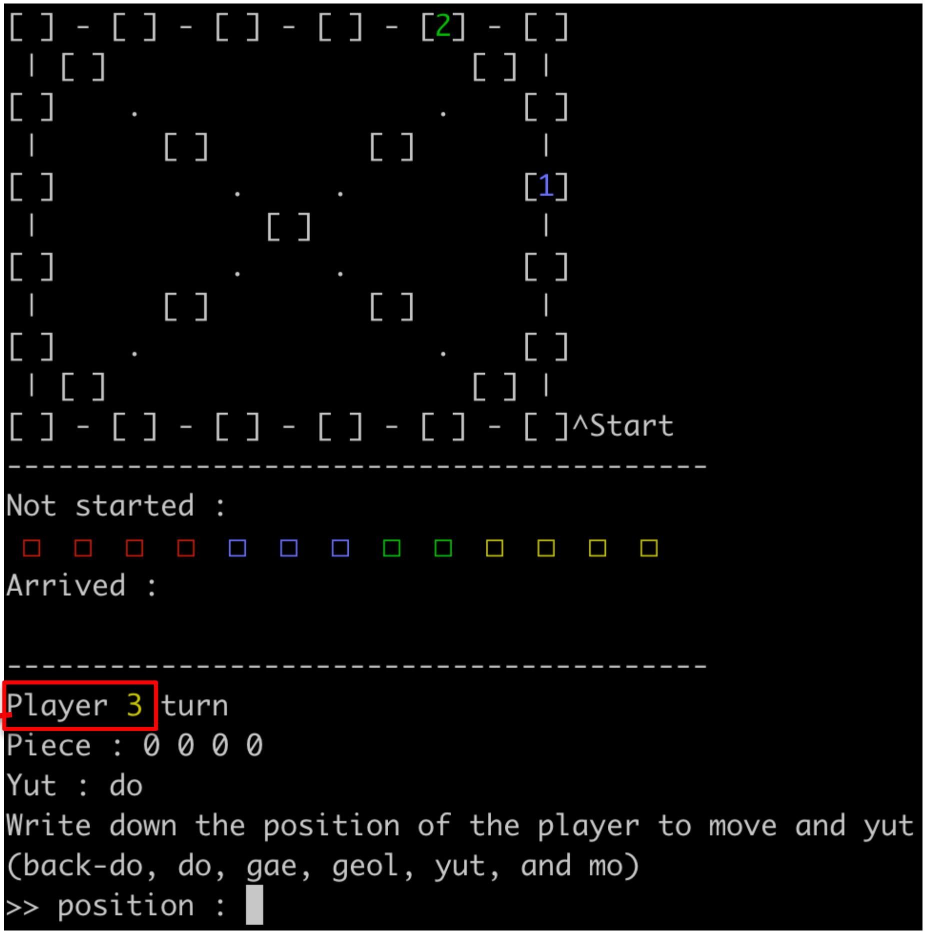
- Game visualization (ex 3)
  - player 2 throws yut sticks and the result is yut

→ throws again and the result is gae

→ moves a new piece with gae (stacking happens)

→ moves pieces at 2 with yut

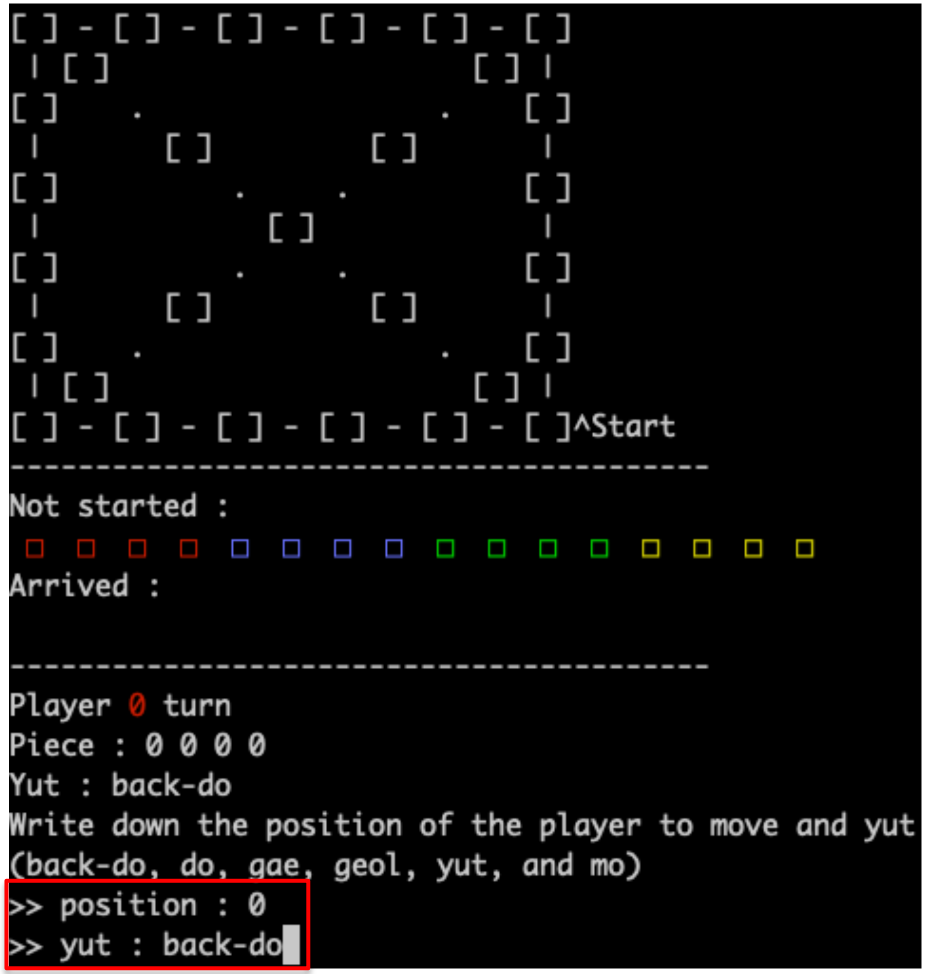
→ Next player's turn



```
[ ] - [ ] - [ ] - [ ] - [2] - [ ]
 | [ ]                   [ ] |
[ ]    .               .    [ ]
 |      [ ]         [ ]      |
[ ]          .   .          [1]
 |            [ ]            |
[ ]          .   .          [ ]
 |      [ ]         [ ]      |
[ ]    .               .    [ ]
 | [ ]                   [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 3 turn
Piece : 0 0 0 0
Yut : do
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position :
```

Gameplay examples

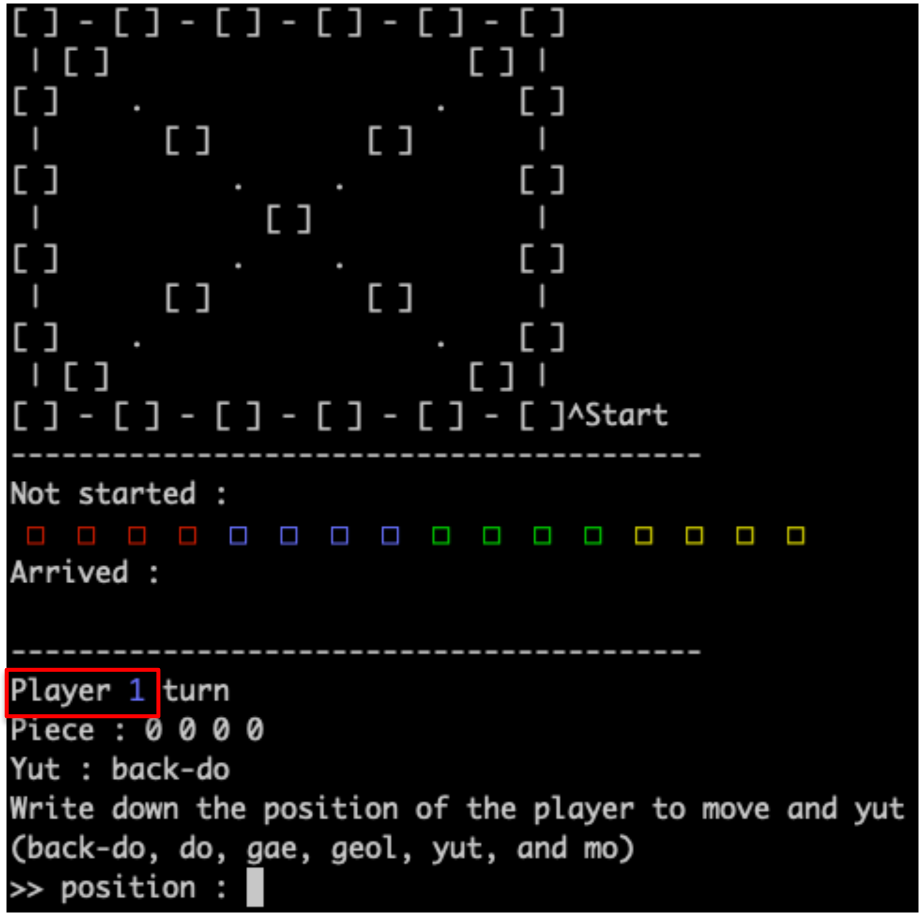
- Game visualization (ex 4)
  - player 0 throws yut sticks and the result is back-do.
    - Since player 0's all pieces have not departed yet, the pieces cannot move.



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                    [ ] |
[ ]    .                .    [ ]
 |      [ ]          [ ]      |
[ ]          .    .          [ ]
 |             [ ]            |
[ ]          .    .          [ ]
 |      [ ]          [ ]      |
[ ]    .                .    [ ]
 | [ ]                    [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 0 turn
Piece : 0 0 0 0
Yut : back-do
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 0
>> yut : back-do
```

Gameplay examples

- Game visualization (ex 4)
  - player 0 throws yut sticks and the result is back-do.
    - Since player 0's all pieces have not departed yet, the pieces cannot move.
  - Nothing happens and it is the next player's turn.



```
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]
 | [ ]                   [ ] |
[ ]    .                 .    [ ]
 |      [ ]         [ ]       |
[ ]          .    .           [ ]
 |             [ ]            |
[ ]          .    .           [ ]
 |      [ ]         [ ]       |
[ ]    .                 .    [ ]
 | [ ]                   [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □  □  □  □  □  □  □  □  □  □  □
Arrived :

-----------------------------------------
Player 1 turn
Piece : 0 0 0 0
Yut : back-do
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position :
```

Machine Intelligence & Data science LAB

Outline

- C++ grammars for Project2
- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures
- Submission and Grading

Basic version rules

- Implement Yunnori so that all the rules described above apply.

- You should allow the number of players and pieces per player to be selectable.
  - 2, 3, or 4

Outline

- C++ grammars for Project2
- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures
- Submission and Grading

Machine Intelligence & Data science LAB

Advanced version rules

- Additional functionalities
    1. Different skills per player
    2. Login
    3. Pausing and saving games
    4. Loading the saved game

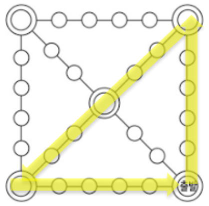
- The number of pieces per player is always 4.

- Otherwise, the rules are the same as in the basic version.

Advanced version rules

- Different skills per player
  - Players should choose one of the following animals:
    - pig, dog, sheep, or cow
  - Pig
    - If someone else (player A) catches you, A ends his turn immediately.
  - Dog
    - Even if the stick result is geol, you should throw again.
  - Sheep
    - If you catch someone else,
      you have two chances to throw again.
  - Cow
    - You can only move along the yellow path.

Obviously, if you get yut or mo, throw again as basic version rules!
(ex1) 1. do / 2. gae
(ex2) 1. gae / 2. yut → mo → geol
(ex3) 1. mo → do / 2. yut → gae



Machine Intelligence & Data science LAB

Advanced version rules

- Different skills per player
  - Players should choose one of the following animals:
    - pig, dog, sheep, or cow

  - F

tely.

  - D

  - S

Caution!!

What happens if **Sheep** catches **Pig**?

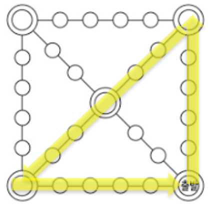
Pig is stronger than Sheep.

So Sheep cannot use its skill and his turn ends immediately.

If you catch someone else,
you have two chances to throw again.

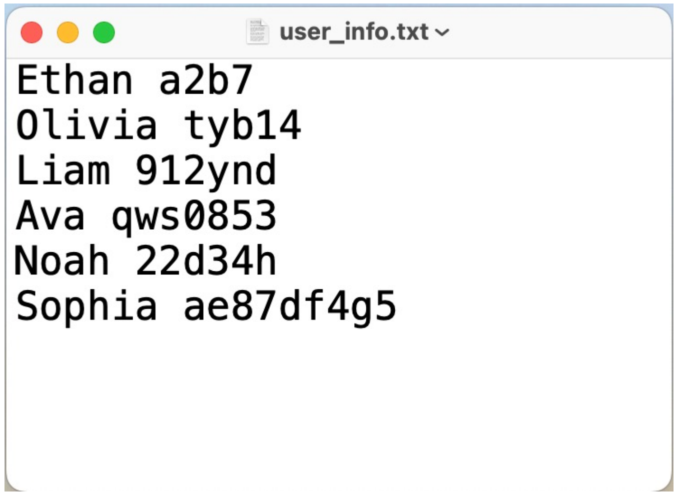
(ex1) 1. do / 2. gae
(ex2) 1. gae / 2. yut → mo → geol
(ex3) 1. mo → do / 2. yut → gae

  - Cow
    - You can only move along the yellow path.



Advanced version rules

- Login
  - ID and password information exists in user_info.txt
    - ID, password information for one player per line
    - ID is English and the password is a mix of alphanumeric characters.



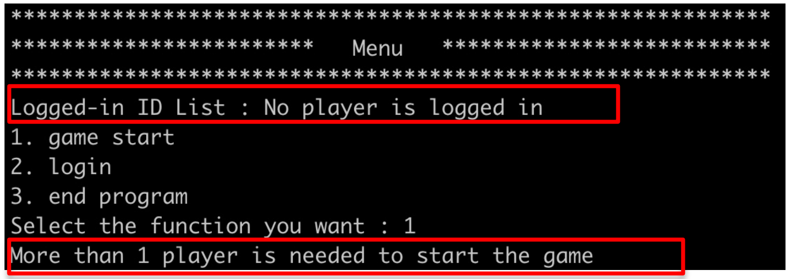
```
Ethan a2b7
Olivia tyb14
Liam 912ynd
Ava qws0853
Noah 22d34h
Sophia ae87df4g5
```

user_info.txt may change at grading time.

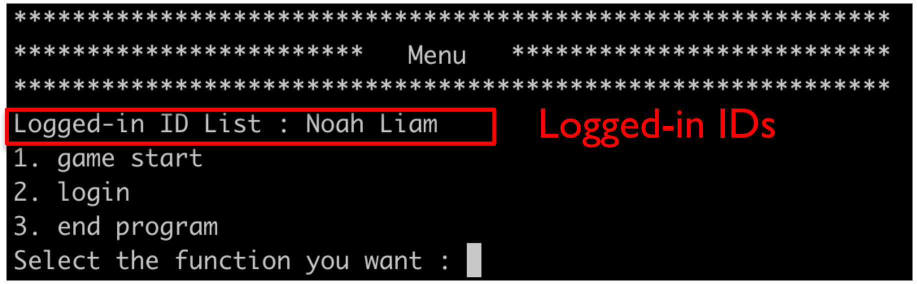
Machine Intelligence & Data science LAB

Advanced version rules

- Login
  - More than one person should login to start the game.



```
****************************************************************
***********************     Menu     ***************************
****************************************************************
Logged-in ID List : No player is logged in
1. game start
2. login
3. end program
Select the function you want : 1
More than 1 player is needed to start the game
```



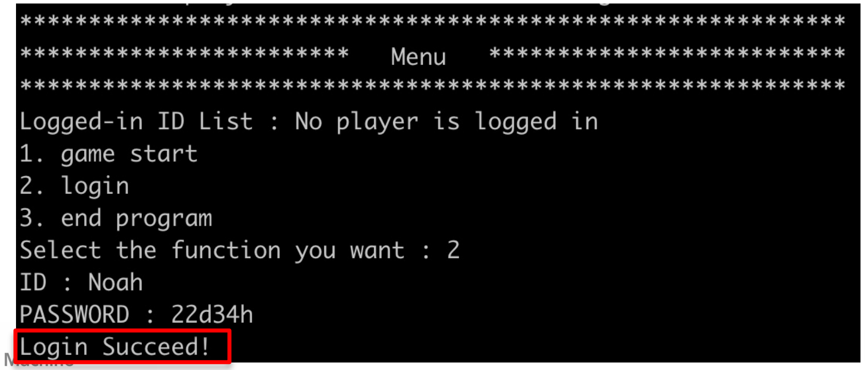
```
****************************************************************
***********************     Menu     ***************************
****************************************************************
Logged-in ID List : Noah Liam
1. game start
2. login
3. end program
Select the function you want :
```

Logged-in IDs

We assume that players who are already logged in will not attempt to login again.

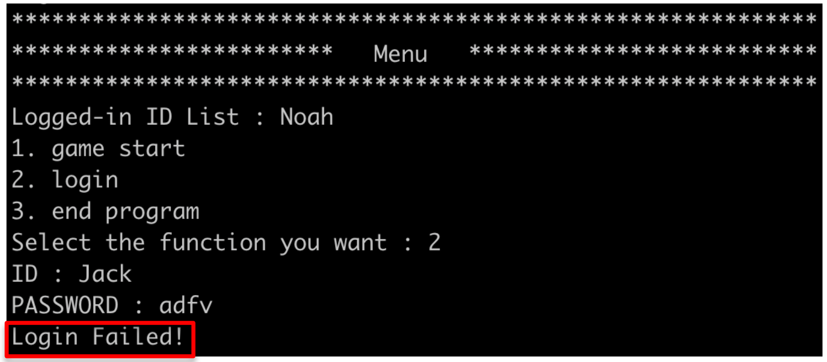
  - Cases of "Login Failed!"

    1. ID does not exist in the user_info.txt / 2. The password is incorrect.



```
****************************************************************
***********************     Menu     ***************************
****************************************************************
Logged-in ID List : No player is logged in
1. game start
2. login
3. end program
Select the function you want : 2
ID : Noah
PASSWORD : 22d34h
Login Succeed!
```

Login succeed

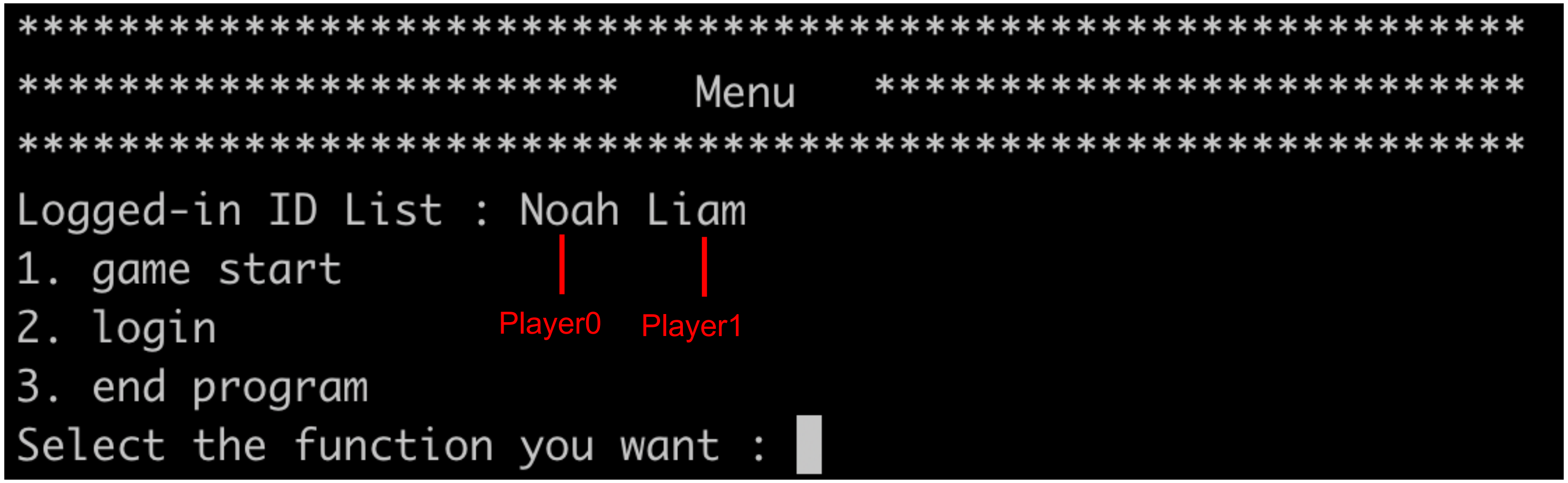


```
****************************************************************
***********************     Menu     ***************************
****************************************************************
Logged-in ID List : Noah
1. game start
2. login
3. end program
Select the function you want : 2
ID : Jack
PASSWORD : adfv
Login Failed!
```

Login failed

Advanced version rules

- Login
  - Player number order (Player 0, Player 1, Player 2, Player 3) is the same as login order



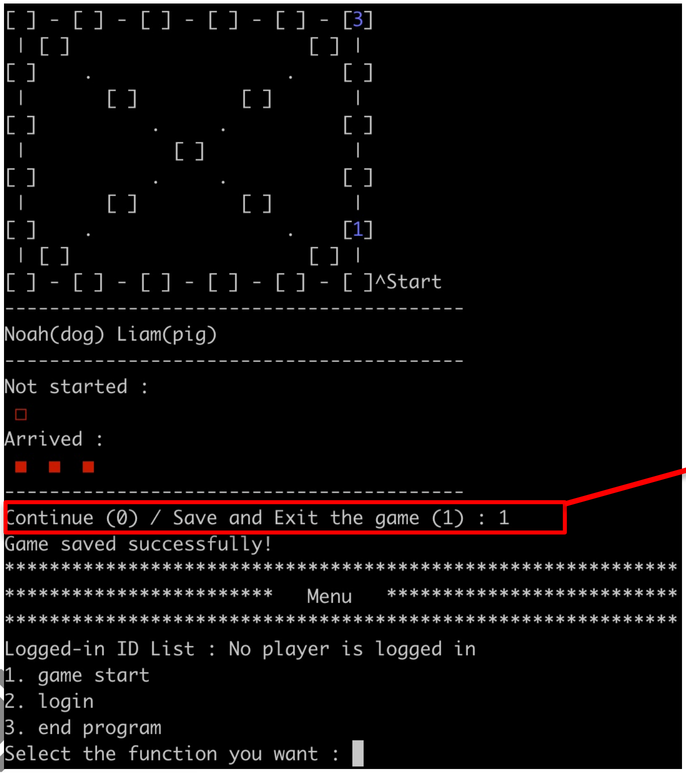
```
************************************************************
************************     Menu     **********************
************************************************************
Logged-in ID List : Noah Liam
1. game start
2. login
3. end program
Select the function you want : 
```

If there exists a saved game with player order Liam(Player0) Noah(Player1), then Liam will be Player0, Noah will be Player1.
You can understand what this means after you read the subsequent slides. (p52)

Advanced version rules

- Pausing and saving games
  - If you want to save the current game information and stop in the middle of a game,

→ Save game information in game_info.txt



```
[ ] - [ ] - [ ] - [ ] - [ ] - [3]
 | [ ]                    [ ] |
[ ]    .               .    [ ]
 |       [ ]        [ ]       |
[ ]           .    .        [ ]
 |             [ ]            |
[ ]           .    .        [ ]
 |       [ ]        [ ]       |
[ ]    .               .    [1]
 | [ ]                    [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Noah(dog) Liam(pig)
-----------------------------------------
Not started :
 □
Arrived :
 ■  ■  ■
-----------------------------------------
Continue (0) / Save and Exit the game (1) : 1
Game saved successfully!
************************************************************
************************    Menu    *************************
************************************************************
Logged-in ID List : No player is logged in
1. game start
2. login
3. end program
Select the function you want : 
```

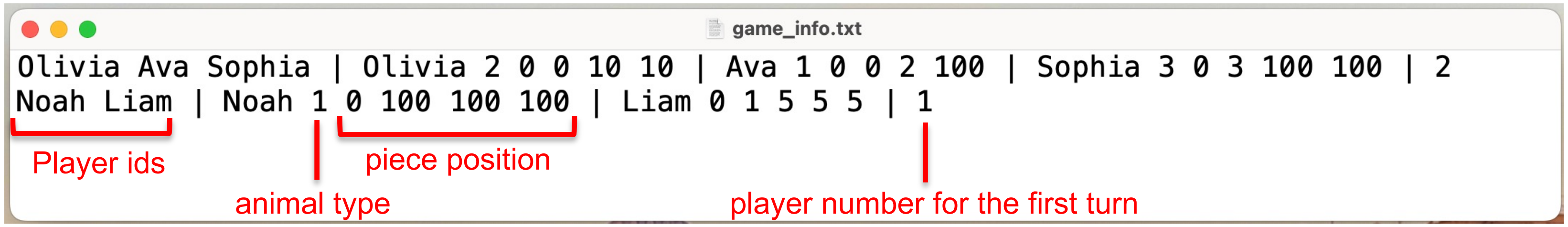
It should be asked between players' turns.

After saving, go back to the beginning of the game. (Initial screen with no one logged in).

Advanced version rules

- Pausing and saving games
  - Multiple game information can be stored. (one per line)
  - Saved ID order is the same as player number order

<game_info.txt>



```
Olivia Ava Sophia | Olivia 2 0 0 10 10 | Ava 1 0 0 2 100 | Sophia 3 0 3 100 100 | 2
Noah Liam | Noah 1 0 100 100 100 | Liam 0 1 5 5 5 | 1
```

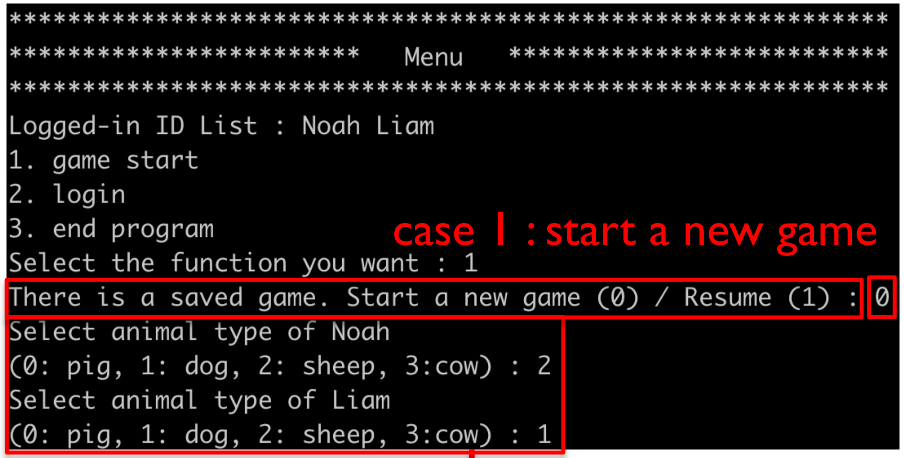
❖ 100 means arrived piece

Machine Intelligence & Data science LAB

Advanced version rules

- Loading the saved game
  – If logged-in IDs before the game starts are the same as that of a saved games stored in game_info.txt,

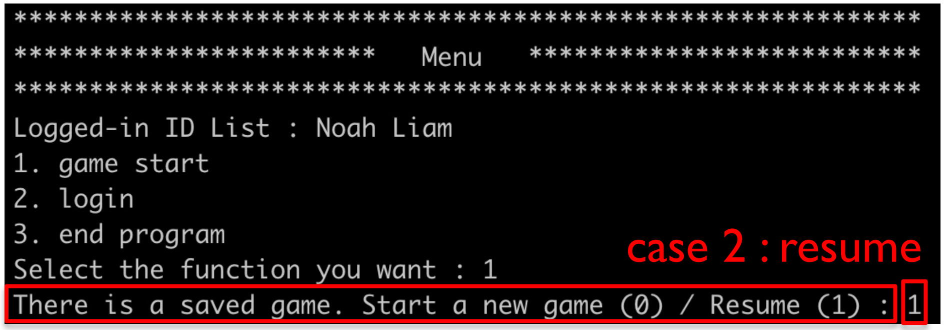
  → Ask whether to load the saved game



```
***************************************************************
***********************    Menu    ***********************
***************************************************************
Logged-in ID List : Noah Liam
1. game start
2. login
3. end program
Select the function you want : 1
There is a saved game. Start a new game (0) / Resume (1) : 0
Select animal type of Noah
(0: pig, 1: dog, 2: sheep, 3:cow) : 2
Select animal type of Liam
(0: pig, 1: dog, 2: sheep, 3:cow) : 1
```

case 1 : start a new game

then a new game starts!!



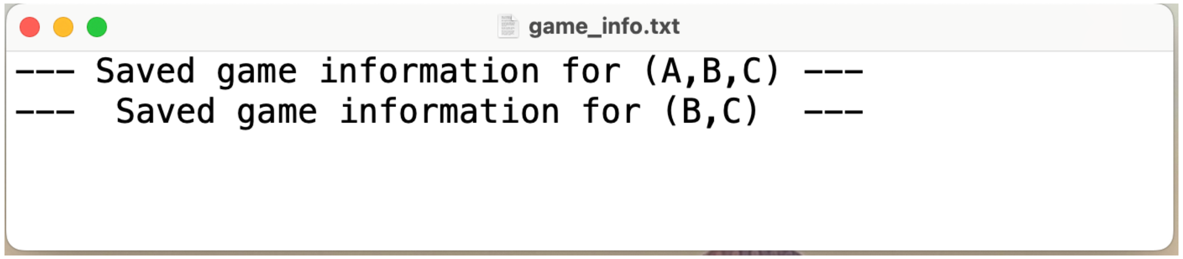
```
***************************************************************
***********************    Menu    ***********************
***************************************************************
Logged-in ID List : Noah Liam
1. game start
2. login
3. end program
Select the function you want : 1
There is a saved game. Start a new game (0) / Resume (1) : 1
```

case 2 : resume

The saved game starts!!

Advanced version rules

- Loading the saved game
  - Only the logged-in ID combination has to match (regardless of login order).
  - The player number order is the same as the saved order.


game_info.txt

```
--- Saved game information for (A,B,C) ---
---  Saved game information for (B,C)  ---
```

❖ Login ID combinations must be exactly the same.
- When B, C, D log in, you should start a new game.

❖ You should play with the saved order, not the newly logged-in ID order.
- If logged-in in the order B, A, C and then resuming the saved game, the player number order should be A,B,C.

Machine Intelligence & Data science LAB

Outline

- C++ grammars for Project2
- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures
- Submission and Grading

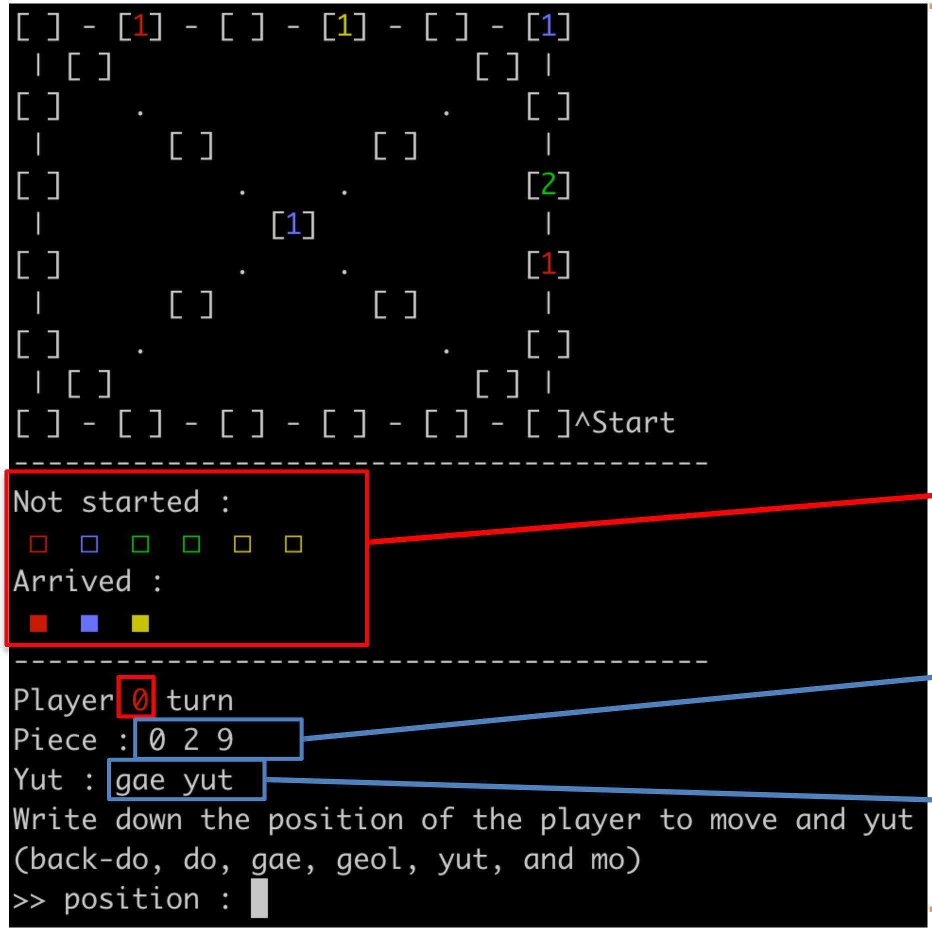
Machine Intelligence & Data science LAB

Visualization rules

- You should print the game board, tokens, game progress, players' states, etc. to the terminal window.
  - See p30-p32.

- It should match our visualization result perfectly.
  1. There are no blank lines (=lines with no text).
  2. To avoid mistakes due to minor differences such as spacing, we will provide the output text in the skeleton-code files.
     - Copy and use it for your implementation.

Visualization rules



```
[ ] - [1] - [ ] - [1] - [ ] - [1]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |      [ ]           [ ]      |
[ ]          .     .          [2]
 |              [1]            |
[ ]          .     .          [1]
 |      [ ]           [ ]      |
[ ]    .                 .    [ ]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Not started :
 □  □  □  □  □  □
Arrived :
 ■  ■  ■
-----------------------------------------
Player 0 turn
Piece : 0 2 9
Yut : gae yut
Write down the position of the player to move and yut
(back-do, do, gae, geol, yut, and mo)
>> position : 
```

There are no blank lines in the output!

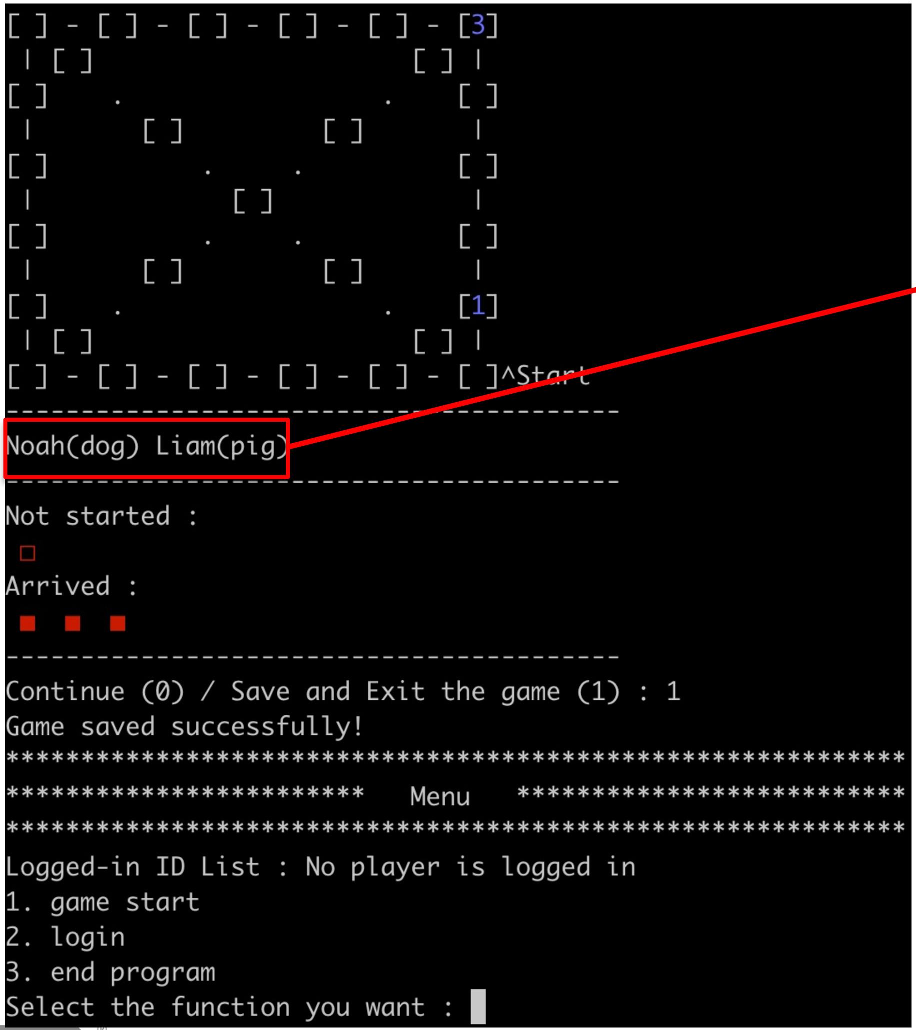
Token colours
- player0 : red
- player1 : blue
- player2 : green
- player3 : yellow

Print in ascending order
(exclude arrived pieces)

Print in the order of
back-do, do, gae, geol, yut, mo

Machine Intelligence & Data science LAB

Visualization rules (Advanced version)



```
[ ] - [ ] - [ ] - [ ] - [ ] - [3]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |       [ ]         [ ]       |
[ ]          .     .          [ ]
 |             [ ]             |
[ ]          .     .          [ ]
 |       [ ]         [ ]       |
[ ]    .                 .    [1]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
-----------------------------------------
Noah(dog) Liam(pig)
-----------------------------------------
Not started :
 □
Arrived :
 ■  ■  ■
-----------------------------------------
Continue (0) / Save and Exit the game (1) : 1
Game saved successfully!
************************************************************
************************   Menu   **************************
************************************************************
Logged-in ID List : No player is logged in
1. game start
2. login
3. end program
Select the function you want :
```

Each player's animal type

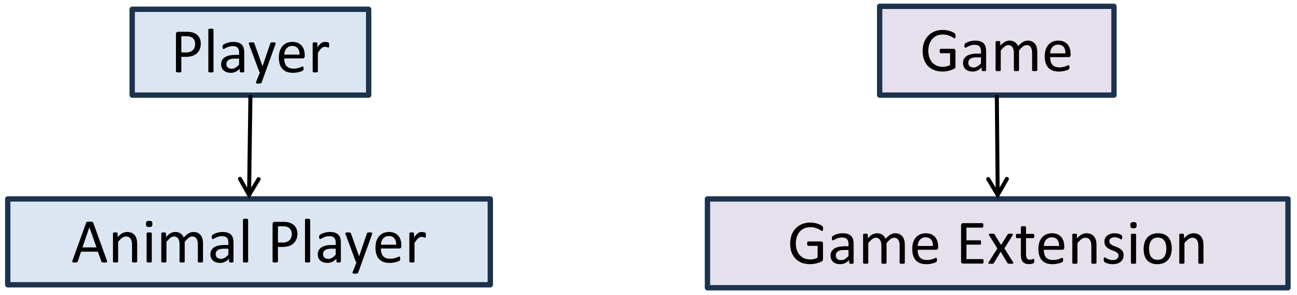
Outline

- C++ grammars for Project2
- Awesome Yunnori
  - Introduction to the gameplay
  - Basic version rules
  - Advanced version rules
  - Visualization rules
  - Code structures
- Submission and Grading

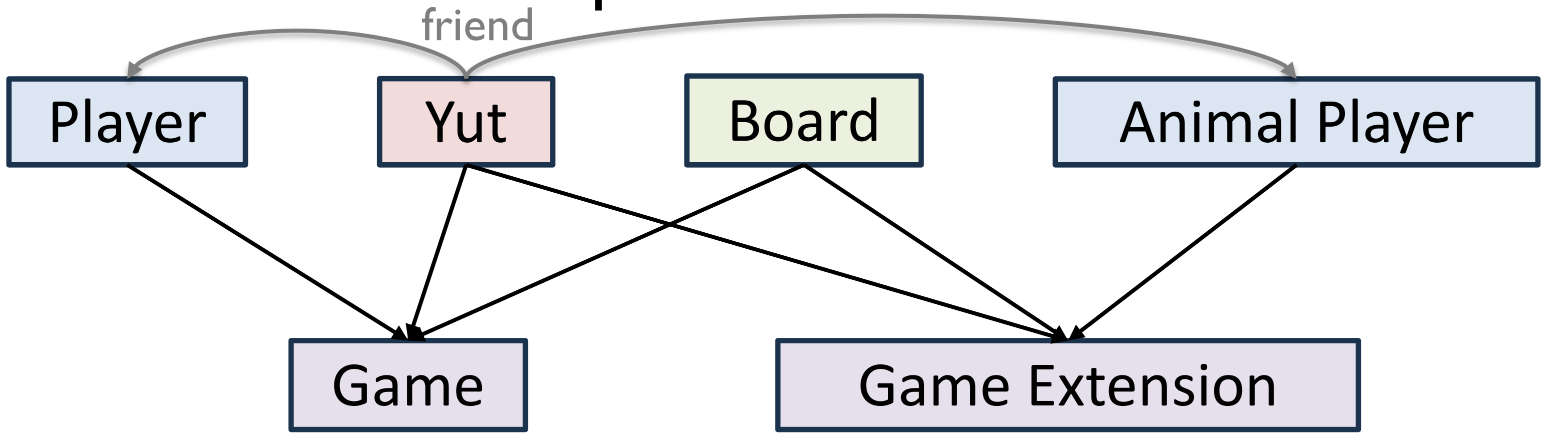
Machine Intelligence & Data science LAB

Code structures

- Class inheritance relationship

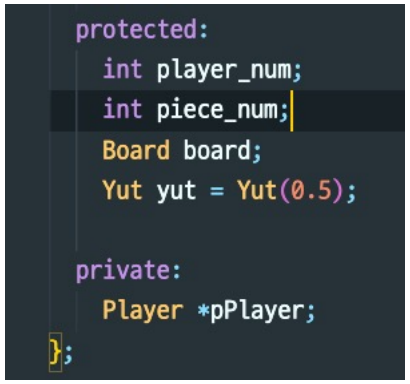


- Declaration relationship



Code structures

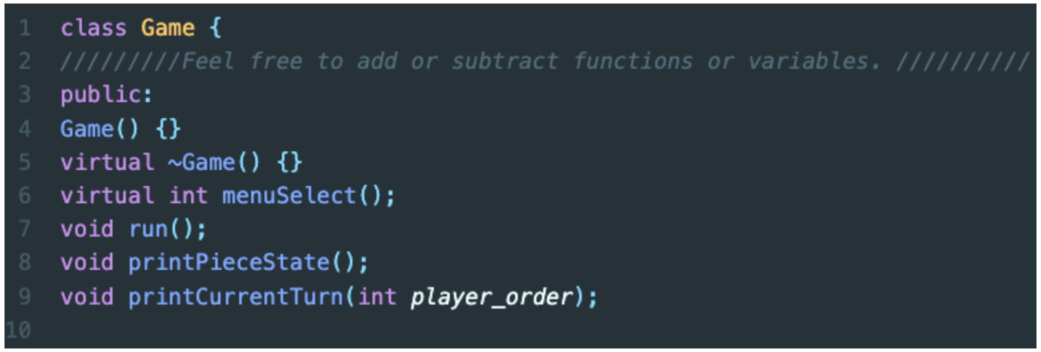
- class Game
  - int player_num
    - Number of players
  - int piece_num
    - Number of pieces per player
  - Board board and Yut yut
    - Board class
    - Yut class
  - Player *pPlayer
    - Object of game player



```
protected:
    int player_num;
    int piece_num;
    Board board;
    Yut yut = Yut(0.5);

private:
    Player *pPlayer;
};
```

Code structures

- class Game
  - int menuSelect();
    - select the number of players and the number of pieces per player.
    - The function returns 0 if the user selects the end program, and returns 1 if the user selects the game start.
  - void run();
    - run the game until the game ends.
    - The function is called when the user selects the game start.

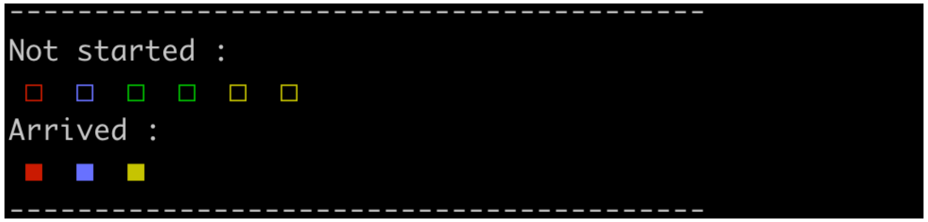


```
class Game {
//////////Feel free to add or subtract functions or variables. //////////
public:
Game() {}
virtual ~Game() {}
virtual int menuSelect();
void run();
void printPieceState();
void printCurrentTurn(int player_order);

```

Machine Intelligence & Data science LAB

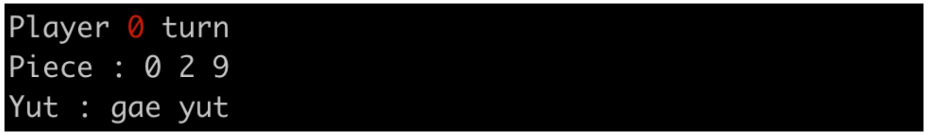
Code structures

- class Game
  - void printPieceState();
    - print the state of the pieces that have not started and the pieces that have arrived.



```
-----------------------------------------
Not started :
 □  □  □  □  □  □
Arrived :
 ■  ■  ■
-----------------------------------------
```

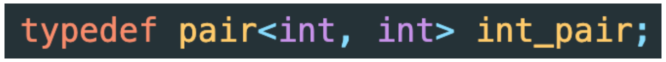
  - void printCurrentTurn();
    - display the player's turn, the position of current pieces, and the current yut list.



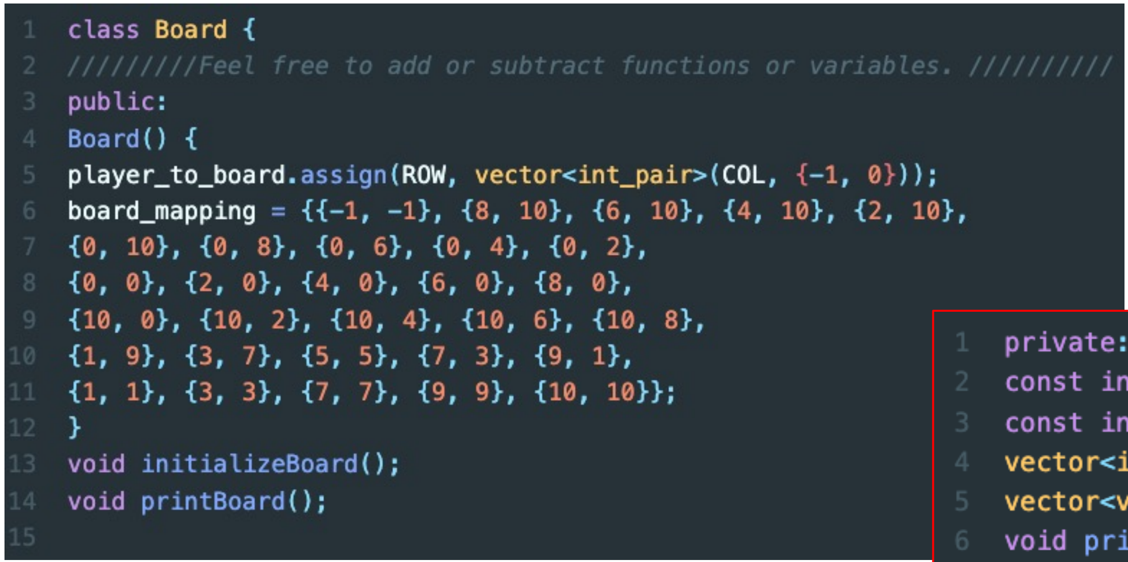
```
Player 0 turn
Piece : 0 2 9
Yut : gae yut
```

Code structures

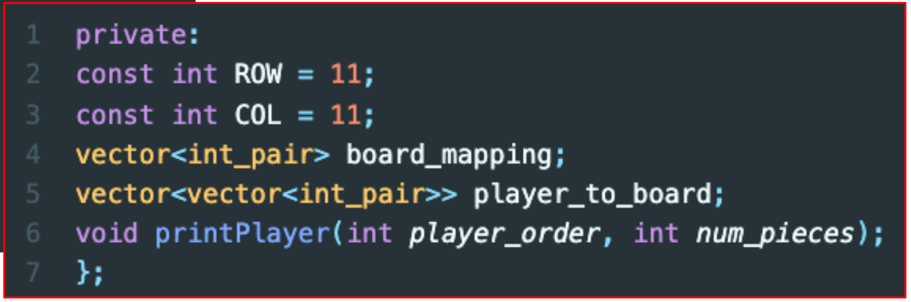
- class Board
  - vector<int_pair> board_mapping
    - The (x,y) coordinates of the position of the piece on the board
  - vector<vector<int_pair> player_to_board
    - Store (which player, number of pieces) values at each board position



```
typedef pair<int, int> int_pair;
```



```
class Board {
//////////Feel free to add or subtract functions or variables. //////////
public:
Board() {
player_to_board.assign(ROW, vector<int_pair>(COL, {-1, 0}));
board_mapping = {{-1, -1}, {8, 10}, {6, 10}, {4, 10}, {2, 10},
{0, 10}, {0, 8}, {0, 6}, {0, 4}, {0, 2},
{0, 0}, {2, 0}, {4, 0}, {6, 0}, {8, 0},
{10, 0}, {10, 2}, {10, 4}, {10, 6}, {10, 8},
{1, 9}, {3, 7}, {5, 5}, {7, 3}, {9, 1},
{1, 1}, {3, 3}, {7, 7}, {9, 9}, {10, 10}};
}
void initializeBoard();
void printBoard();

```



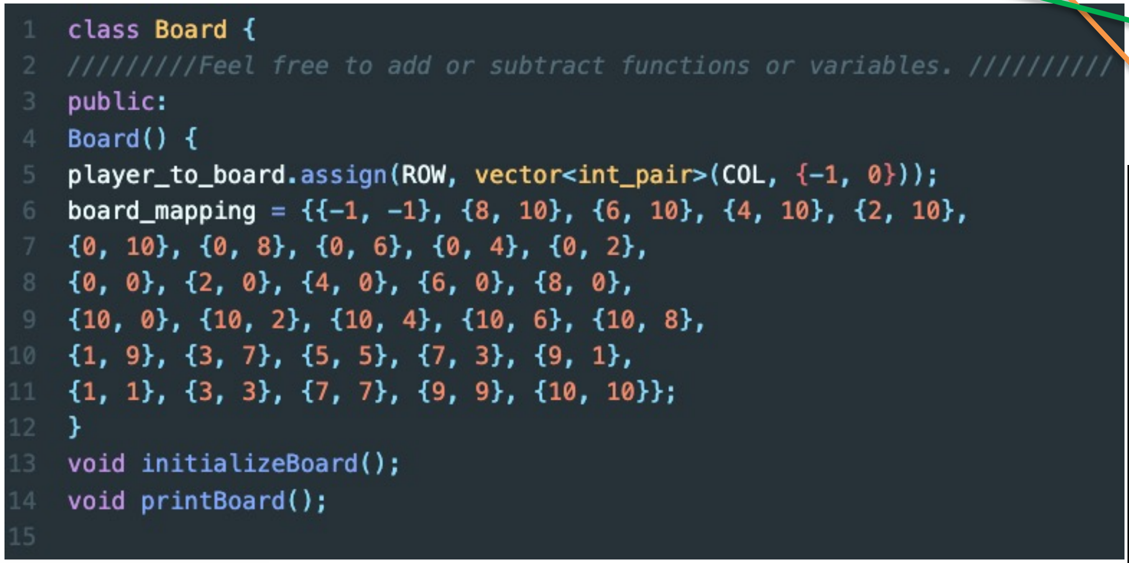
```
private:
const int ROW = 11;
const int COL = 11;
vector<int_pair> board_mapping;
vector<vector<int_pair>> player_to_board;
void printPlayer(int player_order, int num_pieces);
};
```

Machine Intelligence & Data science LAB

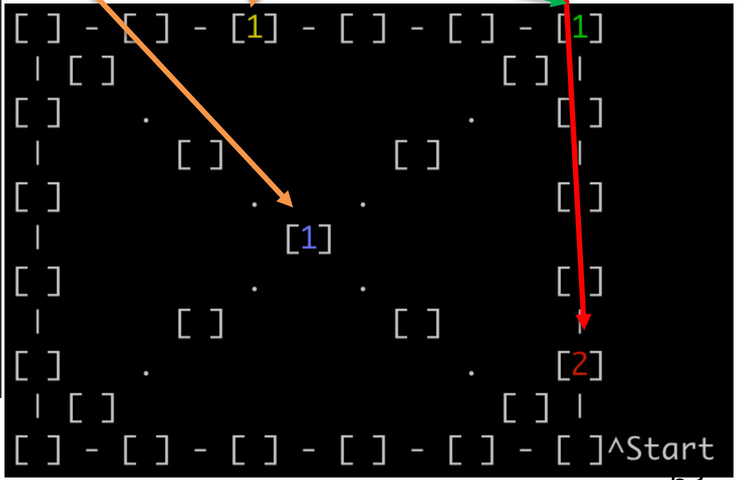
Code structures

- class Board
  - vector<int_pair> board_mapping
    - (ex) board_mapping[22] = {5, 5}, board_mapping[8] = {0, 4}
  - vector<vector<int_pair> player_to_board
    - (ex) player_to_board[0][10] = {2, 1}, player_to_board[8][10] = {0, 2}

```
typedef pair<int, int> int_pair;
```



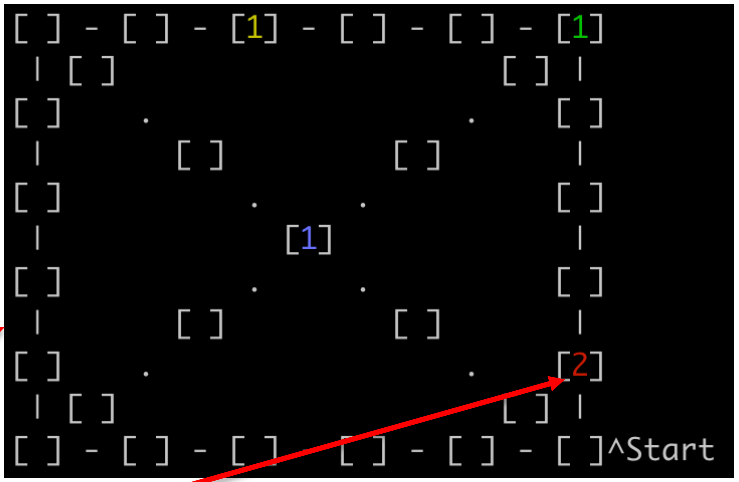
```
class Board {
//////////Feel free to add or subtract functions or variables. //////////
public:
Board() {
player_to_board.assign(ROW, vector<int_pair>(COL, {-1, 0}));
board_mapping = {{-1, -1}, {8, 10}, {6, 10}, {4, 10}, {2, 10},
{0, 10}, {0, 8}, {0, 6}, {0, 4}, {0, 2},
{0, 0}, {2, 0}, {4, 0}, {6, 0}, {8, 0},
{10, 0}, {10, 2}, {10, 4}, {10, 6}, {10, 8},
{1, 9}, {3, 7}, {5, 5}, {7, 3}, {9, 1},
{1, 1}, {3, 3}, {7, 7}, {9, 9}, {10, 10}};
}
void initializeBoard();
void printBoard();
```



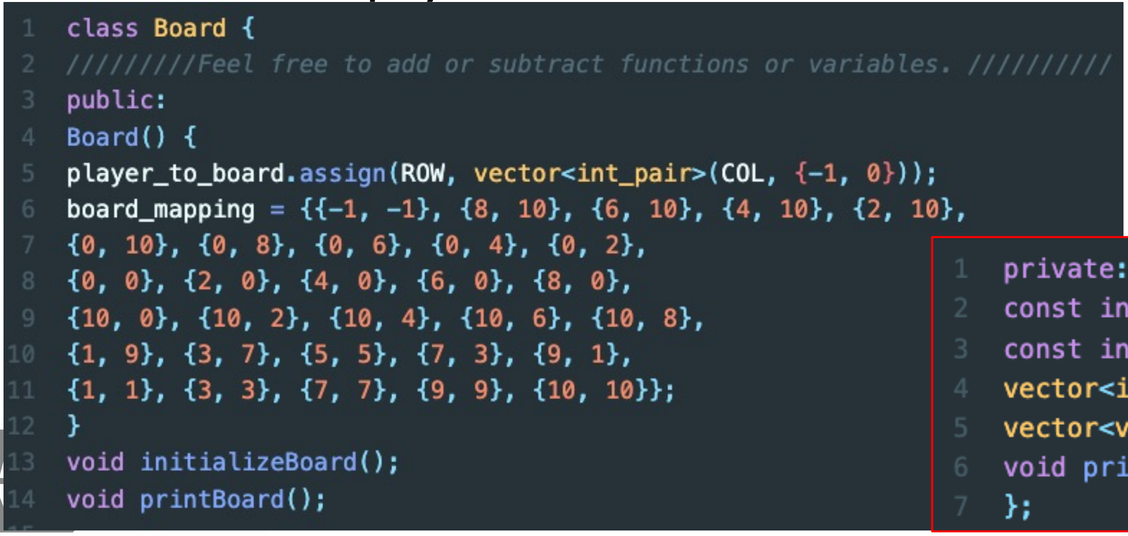
```
[ ] - [ ] - [1] - [ ] - [ ] - [1]
 | [ ]                    [ ] |
[ ]    .               .    [ ]
 |       [ ]         [ ]      |
[ ]          .     .        [ ]
 |              [1]           |
[ ]          .     .        [ ]
 |       [ ]         [ ]      |
[ ]    .               .    [2]
 | [ ]                    [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
```

Code structures

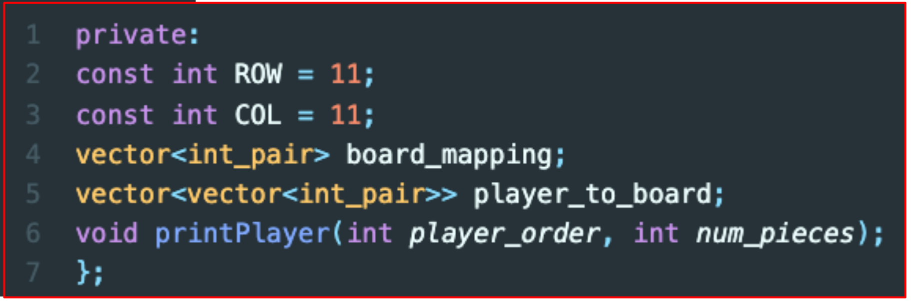
- class Board
  - void initializeBoard();
    - Initialize board state
  - void printBoard();
    - Print the entire board
  - void printPlayer
    - Print player in the board



```
[ ] - [ ] - [1] - [ ] - [ ] - [1]
 | [ ]                     [ ] |
[ ]    .                 .    [ ]
 |       [ ]         [ ]       |
[ ]          .     .          [ ]
 |             [1]             |
[ ]          .     .          [ ]
 |       [ ]         [ ]       |
[ ]    .                 .    [2]
 | [ ]                     [ ] |
[ ] - [ ] - [ ] - [ ] - [ ] - [ ]^Start
```



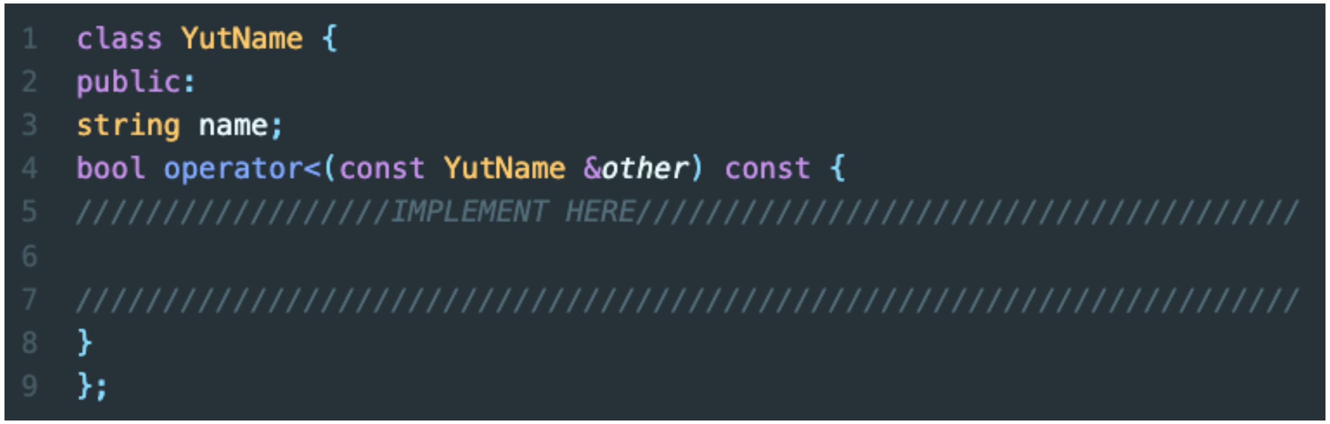
```cpp
class Board {
//////////Feel free to add or subtract functions or variables. //////////
public:
Board() {
player_to_board.assign(ROW, vector<int_pair>(COL, {-1, 0}));
board_mapping = {{-1, -1}, {8, 10}, {6, 10}, {4, 10}, {2, 10},
{0, 10}, {0, 8}, {0, 6}, {0, 4}, {0, 2},
{0, 0}, {2, 0}, {4, 0}, {6, 0}, {8, 0},
{10, 0}, {10, 2}, {10, 4}, {10, 6}, {10, 8},
{1, 9}, {3, 7}, {5, 5}, {7, 3}, {9, 1},
{1, 1}, {3, 3}, {7, 7}, {9, 9}, {10, 10}};
}
void initializeBoard();
void printBoard();
```



```cpp
private:
const int ROW = 11;
const int COL = 11;
vector<int_pair> board_mapping;
vector<vector<int_pair>> player_to_board;
void printPlayer(int player_order, int num_pieces);
};
```

Code structures

- class YutName
  - string name: back-do, do, gae, geol, yut, and mo
  - bool operator< : to print a list of yut results in the order of back-do, do, gae, geol, yut, mo

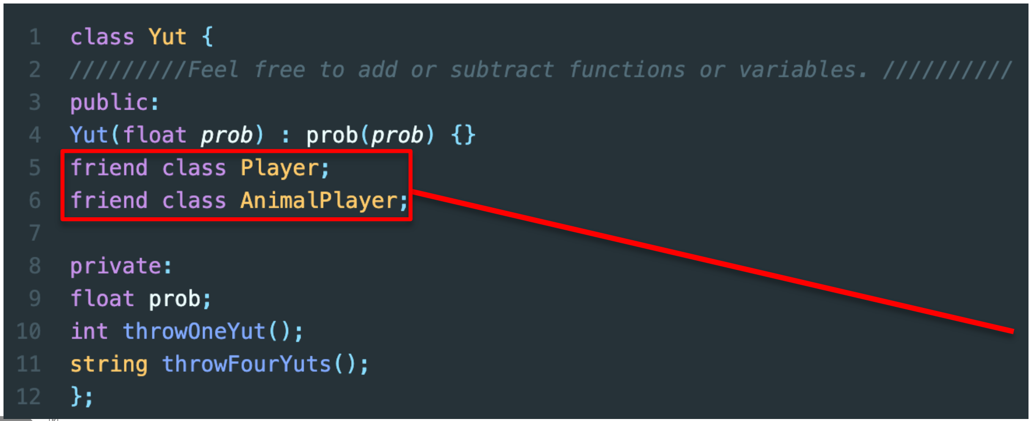


```cpp
class YutName {
public:
string name;
bool operator<(const YutName &other) const {
//////////////////IMPLEMENT HERE////////////////////////////////////

////////////////////////////////////////////////////////////////////
}
};
```

Machine Intelligence & Data science LAB

Code structures

- class Yut
  - float prob (=0.5)
    - probability of a single yut stick coming up heads or tails
  - int throwOneYut() : throw a single yut stick (0 or 1)
  - string throwFourYuts() : throw four yut sticks
    - return: back-do, do, gae, geol, yut, mo



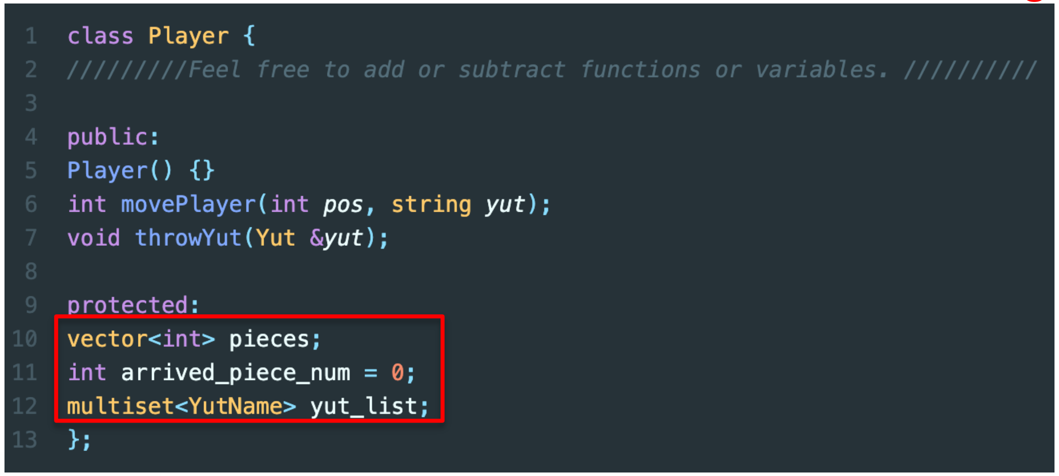
```
class Yut {
//////////Feel free to add or subtract functions or variables. //////////
public:
Yut(float prob) : prob(prob) {}
friend class Player;
friend class AnimalPlayer;

private:
float prob;
int throwOneYut();
string throwFourYuts();
};
```

Friend class
Player class can use private functions in Yut class

Code structures

- class Player
  - vector<int> pieces : a list of the player's pieces positions
  - int arrived_piece_num : number of arrived pieces
  - multiset<Yutname> yut_list: stores the list of yut results that the player currently has.

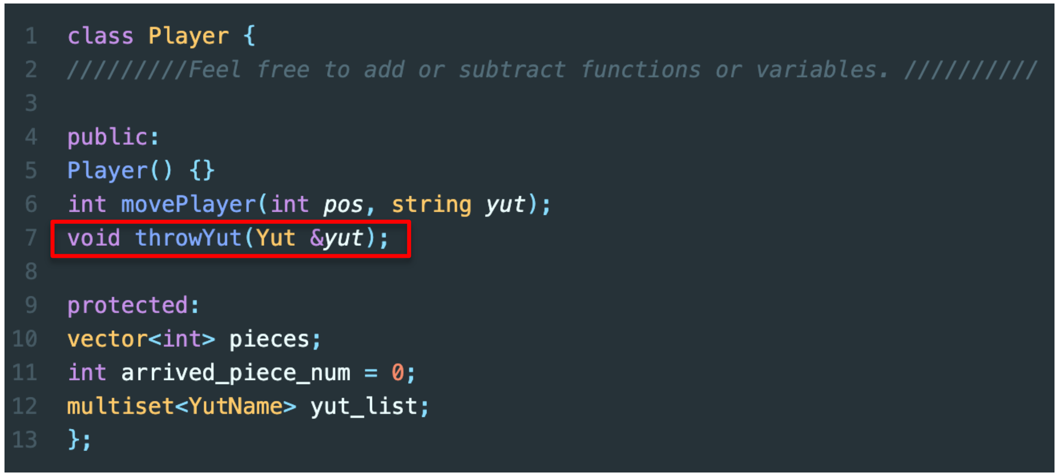
Compared to <std::set>, <std::multiset> allows duplicate elements to be stored while maintaining them in sorted order.



```cpp
class Player {
//////////Feel free to add or subtract functions or variables. //////////

public:
Player() {}
int movePlayer(int pos, string yut);
void throwYut(Yut &yut);

protected:
vector<int> pieces;
int arrived_piece_num = 0;
multiset<YutName> yut_list;
};
```

Code structures

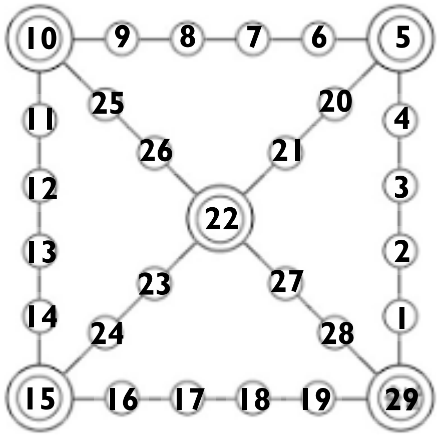
- class Player
  - void throwYut(Yut &yut)
    - Player can throw Yut (Yut class declares Player class as a friend class)

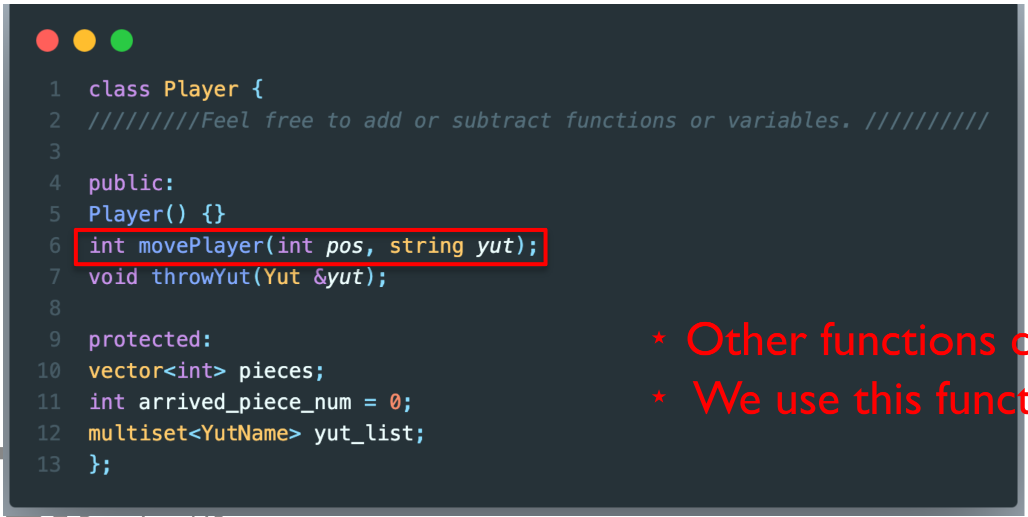


```cpp
class Player {
//////////Feel free to add or subtract functions or variables. //////////

public:
Player() {}
int movePlayer(int pos, string yut);
void throwYut(Yut &yut);

protected:
vector<int> pieces;
int arrived_piece_num = 0;
multiset<YutName> yut_list;
};
```

Code structures

- class Player
  - int movePlayer(int pos, string yut)
    - Input
      - int pos: initial position of piece
      - string yut: yut result (ex. do, gae, geol, …)
    - Return: the final position of the piece
      - if a piece is arrived, then return 100





```
class Player {
//////////Feel free to add or subtract functions or variables. //////////

public:
Player() {}
int movePlayer(int pos, string yut);
void throwYut(Yut &yut);

protected:
vector<int> pieces;
int arrived_piece_num = 0;
multiset<YutName> yut_list;
};
```

⋆ Other functions cannot be called within movePlayer().

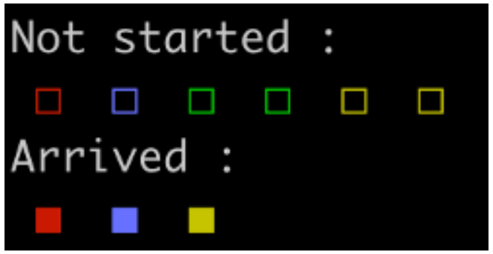
⋆ We use this function for grading.

Implementation

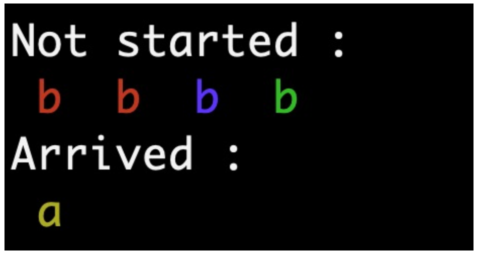
- Feel free to implement the rest of the functions you need.
- The way functions are declared in the skeleton can also be changed.
  - You may need to modify them to implement advanced version game. (e.g. using virtual functions...)
- You can include other libraries (as long as c++17 allows).
- You can declare functions or variables as needed.
- You can add a virtual keyword to the function.

Implementation

- For Window users, '□' and '■' are not printed correctly in the terminal window.
  - Replace them with 'a' and 'b' to run your codes, and change them back to '□' and '■' when you submit your final codes.



```
Not started :
 □  □  □  □  □  □
Arrived :
 ■  ■  ■
```



```
Not started :
 b  b  b  b
Arrived :
 a
```

- The demo is available in Elice.
  - However, be aware that due to a problem with Elice, it does not allow you to save a new game to game_info.txt.

Machine Intelligence & Data science LAB

Grading

- Basic version
  - The piece moves exactly according to the yut sticks result (14)
  - Catching, stacking (each 4, total 8)
  - Throwing yut and using multiset<YutName> (10)
  - Game-end based on the win condition (3)
  - Print the board and player state (10)
- Advanced version
  - Login (10)
  - Pause, save, and load the game (15)
  - Animal piece function (each 5, total 20)
- Check for other errors and visualization rules (10)

Grading

- You have to consider the terminal output format.

- Assume there are only valid inputs.
  - No need to throw exceptions for disallowed input

- Your code will be graded on the Elice platform.

❖ Note that elice is limited to 30 minutes of execution time

- DO NOT COPY OR CHEAT (Plagiarism = F grade)

Submission

- File structure
  - Put all code (game, game_extension, player, yut, board .cpp/.hpp) into a directory named "20XX-XXXXX_name_project2" (you do not need to submit main.cpp, simulater.cpp/.hpp)
  - Then zip it into a 20XX-XXXXX_name_project2.zip
    - (ex) 2024-12345_김프방_project2.zip

- Submit a zip file which includes source codes to Elice
  - Due Date 06/09(Sun) 11:59PM
    - For each day after deadline you score will be deducted by 20%

Machine Intelligence & Data science LAB