Assessment

Tingyi Wang (uni: tw2677)

Part1

1A

Use case1: start game

Actor: player1

Description: player1 wants to start a new game

Basic flow: player1 enters the endpoint /newgame first.

Branch 1: click start game without choosing style. The game won't start and a window shows up that ask player1 to select one option.

Branch 2: click 'X' and then click start game. The game starts and player1 holds 'X' for his or her move. And player2 holds 'O' automatically. A link for player2 appears.

Branch 3: click 'O' and then click start game. The game starts and player1 holds 'O' for his or her move. And player2 holds 'X' automatically. A link for player2 appears.

Use case2: join game

Actor: player1, player2

Description: player1 has started the game and want player2 to join.

Basic flow: player1 enters the endpoint /newgame first. Then player1 chooses 'X' and clicks start game. A link for player2 appears. Player1 sends the link to player2. Player2 opens the link in his or her browser. Player1 and player2 see the same empty game board. A message below player1's board says it's now player1's turn.

Branch 1: player1 click on the game board. A 'X' appears on the grid where player1 clicks.

Branch 2: player2 click on the game board. Nothing appears on the grid.

Use case3: make a move

Actor: player1, player2

Description: player1 has started the game and player2 has successfully join the game. And player 1 makes the first move. Now the game should continue.

Basic flow: player1 enters the endpoint /newgame first. Then player1 chooses 'X' and clicks start game. A link for player2 appears. Player1 sends the link to player2. Player2 opens the link in his or her browser. Player1 makes the first move at (0, 0).

Branch 1: player1 clicks on the game board again. Nothing appears and a message below player1's board says "Please wait for player2".

Branch 2: player2 clicks on the game board at (0, 0). The game board won't change and a message below player2's board says "Invalid move. Please click again." because player1 has already made a move there.

Branch 3: player2 click on the game board except (0, 0), e.g. (1, 0). A 'O' appears at (1, 0) on the game board of player1's and player2's at the same time. The message below both sides' game board suggests next move is for player1.

Use case4: player1 wins

Actor: player1, player2

Description: During the game, player1 who holds 'X' wins the game first. That is, three 'X' combine to a straight line before 'O'. A message appears below both side's game board says "Game Over! Player1 has won the game!".

Basic flow: players still try to click even after player1 has won.

Branch 1: player1 clicks after he or she has won. Nothing changes in UI.

Branch 2: player2 clicks after player1 has won. Nothing changes in UI.

Use case5: player2 wins

Action: player1, player2

Description: During the game, player2 who holds 'O' wins the game first. That is, three 'O' combine to a straight line before 'X'. A message appears below both side's game board says "Game Over! Player2 has won the game!".

Basic flow: players still try to click even after player2 has won.

Branch 1: player1 clicks after player2 has won. Nothing changes in UI.

Branch 2: player2 clicks after he or she has won. Nothing changes in UI.

Use case6: the game is draw

Action: player1, player2

Description: All the grids are clicked and no one wins. A message appears below both side's game board says "This game is a draw!".

Basic flow: players still try to click even after the game comes to a draw.

Branch 1: player1 clicks after the game is draw. Nothing changes in UI.

Branch 2: player2 clicks after the game is draw. Nothing changes in UI.

1B

For use case 1, method(s): setters and getters of class Player and GameBoard.

For use case 2, method(s): setters and getters of class Player and GameBoard.

For use case 3, method(s): conduct from class Move, which is used to update the valid move to the game board; checkwinner from class GameBoard, which is used to check whether there is a winner or the game is draw after this move.

For use case 4, method(s): conduct from class Move, which is used to update the valid move to the game board; checkwinner from class GameBoard, which is used to check whether there is a winner or the game is draw after this move; updatewinner from class GameBoard, which is used to set winner for GameBoard or set the game to draw according to the input value.

For use case 5, method(s): conduct from class Move, which is used to update the valid move to the game board; checkwinner from class GameBoard, which is used to check whether there is a winner or the game is draw after this move; updatewinner from class GameBoard, which is used to set winner for GameBoard or set the game to draw according to the input value.

For use case 6, method(s): conduct from class Move, which is used to update the valid move to the game board; checkwinner from class GameBoard, which is used to check whether there is a winner or the game is draw after this move; updatewinner from class GameBoard, which is used to set winner for GameBoard or set the game to draw according to the input value.

1C
First, I would write the methods with functionality to the models instead of controllers. So that the MVC structure is more obvious for my implementation.
Second, I would use more methods, each method only implements a specific function. Thus, they would be easier to test and adjust, also in order to satisfy “SOLID” software engineering design principles.

Part2

2A
Pay-Attention:

Enable pay_attention

post /users/{userId}/pay_attention
description: enable the pay_attention function
request parameter(s): userId
response: status code

Check eye movement

get /users/{userId}/pay_attention/eye
description: get data from Soli that senses the eye information of user, such as the eyeballs’ movement, frequency of wink, direction the user is looking at, etc.
request parameter(s): userId
response:

```
{
    “eyeMovement”: [
        {
        “id”: xxxx,
        “eyeballMovement”: {xxx, xxx, xxx, xxx, …},
        “frequency”: xxx,
        “direction”: {xxx, xxx, xxx, xxx, …}
        }
    ]
}
```

Check mouth movement

get /users/{userId}/pay_attention/mouth
description: get data from Soli that senses the mouth information of user, such as whether the user is talking, the user is smiling.
request parameter(s): userId
response:

```
{
    “mouthMovement”: [
        {
        “id”: xxxx,
```

```
                "userIsTalking": false,
                "userIsSmiling": false
                }
        ]
}
```

Check facial expression

get /users/{userId}/pay_attention/facial

description: get data from Soli that senses the facial expression of user, such as happy, sad, no expression, staring, wondering, etc.

request parameter(s): userId

response:

```
{
        "facialExpression": [
                {
                "id": xxxx,
                "expression": "starring"
                }
        ]
}
```

Check head movement

get /users/{userId}/pay_attention/head

description: get data from Soli that senses the head movement of user, such as nodding, shaking, the degree of moving, etc.

request parameter(s): userId

response:

```
{
        "headMovement": [
                {
                "id": xxxx,
                "movePattern": "nodding",
                "degreeOfMovement": "slightly"
                }
        ]
}
```

Check hand movement

get /users/{userId}/pay_attention/hand

description: get data from Soli that senses the hands movement of user, such as waving, degree of movement, up and down, ect.

request parameter(s): userId

response:

```
{
```

```
        “handMovement”: [
                {
                “id”: xxxx,
                “handAction”: “waving”,
                “degreeOfMovement”: “slightly”
                }
        ]
  }
```

Check voice

get /users/{userId}/pay_attention/voice

description: get data from microphone that senses the voice information, such as the volumn, emotion, etc.

```
  {
        “voice”: [
                {
                “id”: xxxx,
                “volumn”: “loud”,
                “emotion”: “wonder”
                }
        ]
  }
```

Calculate degree of attention

post /users/{userId}/pay_attention/degreeOfAttention

description: combine the information we get from above endpoint as input of machine learning model, and output the analysis result. If the model is very powerful, it might even know the user is answering question, taking notes or just absence of mind.

request parameter(s): userId

request body:

```
  {
        “eyeMovement”: [
                {
                “id”: xxxx,
                “eyeballMovement”: {xxx, xxx, xxx, xxx, …},
                “frequency”: xxx,
                “direction”: {xxx, xxx, xxx, xxx, …}
                }
        ],
        “mouthMovement”: [
                {
                “id”: xxxx,
                “userIsTalking”: false,
                “userIsSmiling”: false
```

```
            }
        ],
        “facialExpression”: [
            {
            “id”: xxxx,
            “expression”: “starring”
            }
        ],
        “headMovement”: [
            {
            “id”: xxxx,
            “movePattern”: “nodding”,
            “degreeOfMovement”: “slightly”
            }
        ],
        “handMovement”: [
            {
            “id”: xxxx,
            “handAction”: “waving”,
            “degreeOfMovement”: “slightly”
            }
        ],
        “voice”: [
            {
            “id”: xxxx,
            “volumn”: “loud”,
            “emotion”: “wonder”
            }
        ]
}
response:
{
        “degreeOfAttention”: 95,
        “status”: “answering question”
}
```

2B

User story1:

As a zoom meeting host, I want the API to show me whether attenders are paying high enough attention so that I can adjust my way of talking in order to convey my thoughts efficiently.

User story2:

As a zoom meeting host, I want the API to show me the status of attenders so that I know who are all participating in sharing their ideas.

2C

I would invite some volunteers to use the API and change their eye movement, mouth movement, facial expression, head movement respectively and I would see the response body and result from the endpoint that returns degree of attention. Then it will be easy to check whether these endpoints are working separately.

2D

I would ask a group of volunteers to form a zoom meeting, and ask one of them to be the host. In first scene, the host will be asked to make a long presentation and I will be by his side to record the degreeOfAttention. The value should be lower as time goes by. Then I make the host to make some funny joke, the degree of attention should get higher.

In second scene, the host should give an open question to encourage others to share their ideas just like normal talking. I will record the users' status to see whether the status is correct when the users are talking or listening.