SPRINT 2 - Player and Game classes

1) Summary data	
Team number	1
Sprint technical lead(s)	
Sprint start date	03.03.2021
Sprint end date	16.03.2021

The technical lead may vary from one sprint to the next. This is down to how you collectively organise your team.

2) Individual key contributions	
Team member	**Key contribution(s)**
Henry	Notebook class/UI - pygame
Ben	Game class: buttons, start games and ending games, resolve inactive branches (everybody). Explore pathfinding code possibilities. Update Class and Sequence Diagrams.
Sofia	Human/AI player class
Hayden	Human/AI player class

This data should help you to agree your peer assessment at the end of the project. If there is a dispute over your peer assessment, the markers will refer to this section as evidence to support a final decision.

3) User stories / task cards

Provide text descriptions of any user stories or task cards you have selected for this sprint. These should naturally emerge from the user requirements document and discussion on Canvas. If you produce task cards, they should show the relative priority of the task for this sprint.

Notebook class:
Eventually, and by using the detective cards, players should be able to eliminate all but the true murderer, room and a weapon.

Player class:
When a player reaches a room, they can call out a “suggestion” by calling into that room any other person and any weapon. For instance, if Miss Scarlett was in the lounge, she may call Rev Green and the spanner into that room. She would then suggest that the murder was committed “in the lounge by Rev Green using the spanner”. Spare playing pieces fall under equal suspicion and can be called into rooms by players making a suggestion.
When the suggestion is made, starting from the current player’s left, if they have one of the cards mentioned in the suggestion, then they must show one only of those cards to the current player (unseen by other players). If the player on the left is unable to show a card, then we move to the next player to the left until a player is able to show a card to the current player. Once that happens, or all players have been tried leading back to the current player, then the turn of the current player is ended.

Task cards Player (Human/AI) class:

1. Make a suggestion - have to be in that room
2. Accusation
3. Show cards during someone else's turn

Task cards Game class:

1. Game board
2. Turns
3. How the game starts and ends

Task cards Notebook class:

1. Ability to check, cross or "question mark" any player, room or weapon

4) Requirements analysis

For the user stories/task cards selected, set out what key functional, non-functional and domain requirements you have identified. Remember that functional and non-functional requirements can be further categorised as mandatory ("shall") and desirable ("should"). You can use free text descriptions or tabular formats. Remember that domain requirements cannot be acted upon directly. They require domain expertise to refine them into meaningful functional and non-functional requirements. All requirements should be SMART (Specific, Measurable, Achievable, Realistic and Time-Bounded). The requirements analysis does not need to be exhaustive, but should focus on things that are important for this sprint. They should also form a basis for testing.

Player class:

- The user shall be able to roll dice, and move the corresponding number of places on the game board.
- The user shall have the ability to submit an suggestion
- The user shall have the ability to make a final accusation

Game class:

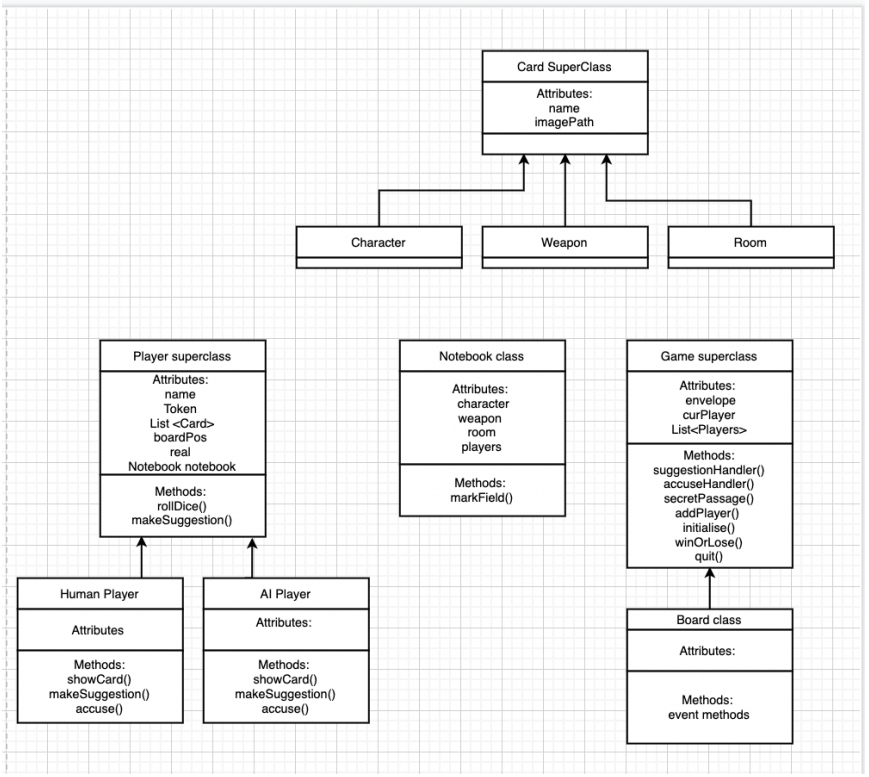
- The player shakes the dice and moves that number of squares along the passage to any room they like.
- Players take turns in a clockwise direction.
- To get the simulation started you will need a means of initialising it with data on the board layout, the players and other data need to make the game function

Notebook class:

- User should be able to always access notebook - no matter who's turn it is
- Ability to add a check, cross or question mark
- "Close and Open" notebook

5) Design

Remember that you only need to do enough design to support the objectives of the sprint. For teams working with OO implementation languages (likely most of you), this would include a class diagram. You may find it useful to develop simple Application Programming Interfaces (APIs) for key classes. This will focus your attention on what each class needs to make available for other classes to use. It also supports good documentation practice and helps coders work together.

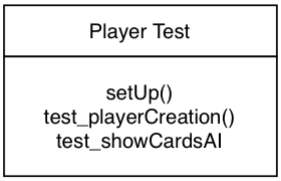


6) Test plan and evidence of testing

You should consider:

- *Unit/component level testing – typically achieved using automated test procedures such as Junit in Java. This level of testing demonstrates that individual classes are working as you intend.*
- *System level testing – typically a human lead and documented test process that shows the prototype working as a whole entity.*

Testing should show that the requirements you set out are being delivered on. They provide a means of showing that we have delivered what the user stores and task cards set out. Remember to identify a useful set of boundary test conditions.
Evidence of testing should demonstrate that the prototype achieved has been tested according to the test plan. If there are deficiencies, then these should be documented, as they will need further work in a subsequent sprint.




```
class TestPlayer(unittest.TestCase):

    def setUp(self):
        self.human = HumanPlayer(cards=[Character.scarlet, Weapon.dagger, Room.diningRoom],
                                 token="hello",
                                 name="Player1",
                                 pos=(0,0),
                                 notebook=0,
                                 )
        self.ai = aiPlayer(cards=[Character.scarlet, Weapon.dagger, Room.diningRoom],
                           token="hello",
                           name="Player1",
                           pos=(0,0),
                           notebook=0,
                           )

    def test_player_creation(self):
        self.assertEqual(self.human.real, True)  # test whether human player is marked as real
        self.assertEqual(self.ai.real, False)  # test whether ai player is marked as not real

    # def cardChange(self):
        # self.
        # self.assertEquals(p1.cards[0], Character.scarlet)

    def test_showCardsAI(self):
        #ai player has card
        cardsToShow = self.ai.showCards(suggestedCards=[Character.plum, Weapon.dagger, Room.ballRoom])
        self.assertEqual(cardsToShow, Weapon.dagger)
        #ai player does not have card
        cardsToShow = self.ai.showCards(suggestedCards=[Character.plum, Weapon.candleStick, Room.ballRoom])
        self.assertEqual(cardsToShow, 0)

    # def test_dice(self):

if __name__ == '__main__':
    unittest.main()
```

Pathfinding Code Test: Function parameters - (start x-coord, start y-coord, end x-coord, end y-coord).

```
moveCharacter(0, 5, 17, 20)
```

```
+------------------------+
|# ##### ######## #######|
|#######  ######  #######|
|#######  ######  #######|
|######   ######  #######|
|#         #####  #######|
|sxxxxxxx ######   ######|
|###### x ##  ##        #|
|#######x                |
|###### x               #|
|###### x               #|
|### ## x         # ######|
|#      x         ########|
|# #### x         ########|
|###### xxx        #######|
|######   xxxx    ########|
|#####       xxxx   #####|
|######         xxx     #|
|#       # #### # x      |
|        ######## x# ####|
|####     ######  e######|
|######  ########  ######|
|######  ########  ######|
|######  ########  ######|
|#######   ####   #######|
+------------------------+
```

7) Summary of sprint

You should consider and discuss:

- *Did you achieve your objectives for this sprint?*
- *Is there a working prototype?*
- *What went well, and what did not go well? If things did not go well, what have you learned and what will you do differently for the next sprint?*
- *Is there any feedback from the customer?*

In this sprint, we realised that the classes have a lot more overlap than we thought. We all worked on our classes individually and decided we will have some meetings where we work on the overlaps between them. We realised we needed to work more on our design. The sequence diagram helped us think of our methods in a different way and helped make the coding easier. We achieved most of our objectives but for sprint 3 we are going to work on the overlaps between classes and the UI.

Notebook Class/UI (Henry)

Notebook class was developed and now contains the array like structure - as seen in various Clue type games, which should enable the checking and crossing of different players, weapons and rooms. The

different marking, crossing and checking capabilities have also been developed, which will be important when running and testing the notebook class within the command line.

The first UI windows have also been established with a starting menu and background image/grid. Some of the elements are not yet fully functional and pygame has been proven to be a challenging library. The documentation has not always proven helpful and instead is rather complicated or not very detailed. Nonetheless, a certain standard of understanding has been reached and will be carried forward to continue developing new UI windows required for the game.

An important lesson was the communication with others when challenges arose. Working with others within the difficulties in the UI class has proven to be successful and has enabled us to make important progress, even after starting difficulties.