**APPENDIX A: SPRINT CYCLE 3 - cards**

|  |  |
| --- | --- |
| 1. **Summary data** | |
| Team number | 7 |
| Sprint technical lead(s) | Ethan, Sarah |
| Sprint start date | 14/4/21 |
| Sprint end date | 21/4/21 |

|  |  |
| --- | --- |
| 1. **Individual key contributions** | |
| **Team member** | **Key contribution(s)** |
| Ethan | Programmer - mechanics and game logic |
| Sarah | Programmer - GUI |
| Umar | Sprint documentation & JavaDoc Testing |

|  |
| --- |
| 1. **User stories / task cards** |
| * The information about the murder is stored on cards placed in the “murder envelope” and remains a mystery until a player makes the correct accusation. * There are 6 cards representing the persons in the house * There are 6 cards representing the weapons * There are 9 cards representing the rooms shown on the playing board * If less than 6 people play (including autonomous players), those playing pieces and their cards not represented should not be removed from the game. Spare pieces are placed in any room or rooms before the game starts whilst the cards remain in the pack and are subsequently dealt to players. * The room cards are shuffled. The top card is placed unseen into the murder envelope * The weapon cards are shuffled. The top card is placed unseen into the murder envelope. * The person cards are shuffled. The top card is placed unseen into the murder envelope. * The dealer shuffles the 18 remaining cards. The cards are dealt to players one at time clockwise around the table, including the dealer. * Some players may receive more cards than others. This is not an issue. |

|  |
| --- |
| 1. **Requirements analysis** |
| F1 - The game shall be playable for a minimum of 2 players and a maximum for 6 players  F9 - 6 cards representing each person, 6 cards representing each weapon and 9 cards representing each room in the house. All cards will be added to the deck  F18 - the “murder envelope” contains the cards of the killer, room where the crime happened and the weapon used. These 3 cards are decided by random. the murder envelope is then placed on the spot x marked on the bored.  F34 - After F18 is completed, the remaining cards in the deck will be dealt to all players including the dealer. It does not matter is some players end up with more cards  NF1 - The software shall be written in Python to ensure maximum portability across platforms  NF4 - The software design shall accommodate future updates or maintenance through well designed code  F20 - The board graphics shall contain an “X” - the murder envelope shall be located on this and only this area of the board  NF6 - Images can be implemented into the board to increase the visual aspect of the game. This is not necessary for the game to run but would assist the game's visuals. |

|  |
| --- |
| 1. **Design** |
| The following image shows the UML diagram for the card class. It shows the structure of how cards will work in the game. There will be a superclass subsuming cards of all different types.    The next image will show the UML diagram for the card mechanics, involving shuffling the cards, adding and removing from the deck etc… This UML diagram also links into suggesting and accusing players of the murder howver, this won’t be implemented until a later date. So as to not overdo the workload for this sprint. |

|  |
| --- |
| 1. **Test plan and evidence of testing** |
| import unittest  from src.card import Card  class MyTestCase(unittest.TestCase):  def test\_something(self):  self.assertEqual(True, True)  def test\_get\_name(self):  card = Card("card1", 1, "w")  name = card.get\_name()  self.assertEqual(name, "card1")  def test\_get\_name\_null\_name(self):  card = Card("", 1, "w")  name = card.get\_name()  self.assertEqual(name, "")  def test\_get\_symbol(self):  card = Card("card1", 1, "w")  symbol = card.get\_symbol()  self.assertEqual(symbol, "w")  def test\_get\_symbol\_null(self):  card = Card("card1", 1, "")  symbol = card.get\_symbol()  self.assertEqual(symbol, "")  def test\_set\_name(self):  card = Card("card1", 1, "")  card.set\_name("card2")  self.assertEqual(card.get\_name(), "card2")  def test\_set\_name\_null(self):  card = Card("card1", 1, "")  card.set\_name("")  self.assertEqual(card.get\_name(), "")  def test\_get\_id(self):  card = Card("", 1, "")  id = card.get\_card\_id()  self.assertEqual(id, 1)  def test\_set\_id(self):  card = Card("", 1, "")  card.set\_card\_id(10)  self.assertEqual(card.get\_card\_id(), 10)  def test\_set\_symbol(self):  card = Card("", 1, "w")  card.set\_symbol("c")  self.assertEqual(card.get\_symbol(), "c")  if \_\_name\_\_ == '\_\_main\_\_':  unittest.main()  import unittest  from src.boardtoken import Token  from src.card import Card  class MyTestCase(unittest.TestCase):  def test\_token\_init(self):  card = Card("card1", 1, "w")  token = Token(1, 1, card)  self.assertTrue(token)  def test\_set\_position(self):  card = Card("card1", 1, "w")  token = Token(1, 1, card)  token.set\_position(2, 2)  x, y = token.get\_position()  self.assertEqual(x, 2)  self.assertEqual(y, 2)  def test\_get\_position(self):  token = Token(1, 1, card)  x, y = token.get\_position()  self.assertEqual(x, 1)  self.assertEqual(y, 1)  def test\_set\_card(self):  card = Card("card", 1, "w")  card1 = Card("card1", 2, "c")  token = Token(x, y, card)  token.set\_card(card1)  self.assertEqual(token.get\_card(), card1)  def test\_get\_card(self):  card = Card("card1", 1, "w")  token = Token(1, 1, card)  token.set\_card(card)  self.assertEqual(card, token.get\_card()) |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **System Testing** | | | | | | | |
| **ID** | **Req** | **Description** | **Inputs** | **Expected** | **Actual** | **Pass/Fail** | **Action** |
| 1 | F9 | Test that each and every player character is represented by a card | Check the contents of the deck and compare to all card types | Every player, even players that are not active, have associated cards | Every player, even players that are not active, have associated cards | Pass | n/a |
| 2 | F9 | Test that each and every weapon is represented by a card | Check the contents of the deck and compare to all card types | Every weapon has associated cards | Every weapon has associated cards | Pass | n/a |
| 3 | F9 | Test that each and every room is represented by a card | Check the contents of the deck and compare to all card types | Every room has associated cards | Every room has associated cards | Pass | n/a |
| 4 | F9 | Test that all cards are added to the deck | Compare count of deck variable to summed number of weapons, rooms, players | All weapon, room and player cards are located in the deck | All weapon, room and player cards are located in the deck | Pass | n/a |
| 5 | F18 | Check that a player card, room card and weapon card are removed from the deck and added to the murder envelope | Check contents of murder envelope variable | A player card, room card and weapon card are removed from the deck and added to the murder envelope | A player card, room card and weapon card are removed from the deck and added to the murder envelope | Pass | n/a |
| 6 | F18 | Check that the cards in the murder envelope are random, therefore different on each run | Compare contents of murder envelope on different game runs | The cards added to the murder envelope are random each time | The cards added to the murder envelope are random each time | Pass | n/a |
| 7 | F20 | Check that the murder envelope is located on the ‘X’ on the board | compare coordinates of murder envelope to board GUI location - visual inspection | The murder envelope will be located on the board GUI on the ‘X’ | The cards have not been implemented to the board GUI | Fail | Implement the card GUI and link to board |
| 8 | NF6 | Check that the murder envelope is represented by a graphical image on the board | visual inspection | The murder envelope is graphically represented on the board | The murder envelope GUI has not been implemented | Fail | Implement the GUI for the murder envelope |
| 9 | NF4 | Check that the code is written well enough to accommodate future updates and maintenance | visual inspection | The code is written well enough to accommodate future updates and maintenance | The code is written well enough to accommodate future updates and maintenance | Pass | n/a |

|  |
| --- |
| 1. **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?* |