Team Mansaf Tests

5.2.1 Change Input Pipe

Scenario:
A player changes the input pipe of a pump.

User Input: ChangeInputPipe
(Note that in the skeleton program, this will correspond to a number, e.g. for Plumbers this will be "8" on the keyboard.)

System Output:
"Enter the number of the pump."

User Input:
4
(4 is just an example, the player should enter the number of the pump they want to change the input pipe of).

System Output:
"Which pipe do you want to set as the input pipe? (1-3)"

User Input:
2
(*2 is just an example, the player should enter the number of the pipe they want to be the inlet for this specific pump, here we assumed that only 3 pipes are connected to that pump).*

System Output:
"Are you on the pump? Enter 1 if yes, enter 0 if not."

UserInput:
1

Condition Check:
Check if the player is standing on the pump and if the pipe they selected was connected to that pump.

Pump.isOccupied() should return True

Pump.connectedPipes.Contains(Pipe) should return True

System Output:
"Checks done."
changeInputPipe(Pump, Pipe);
 Pump.InPipe = Pipe

5.2.2 Change Output Pipe

Scenario:
A player changes the output pipe of a pump.

User Input: ChangeOutputPipe

System Output:
 "Enter the number of the pump."

User Input:
2
(2 is just an example, the player should enter the number of the pump they want to change the output pipe of).

System Output:
"Which pipe do you want to set as the output pipe? (1-3)"

User Input:
1
(*1 is just an example, the player should enter the number of the pipe they want to be the outlet for this specific pump, here we assumed that only 3 pipes are connected to that pump).*

System Output:
"Are you on the pump? Enter 1 if yes, enter 0 if not."

UserInput:
1

Condition Check:
Check if the player is standing on the pump and if the pipe they selected was connected to that pump.

Pump.isOccupied() should return True
Pump.connectedPipes.Contains(Pipe) should return True

System Output:
“Checks done.”
changeOutputPipe(Pump, Pipe);
 Pump.OutPipe = Pipe

5.2.3 Take Turn

Scenario:
A player initiates their turn to play.

User Input: TakeTurn()

Condition Check:
The game controller has to check if the player’s turn is up next.

System Output:
“Is it your turn to play?”

User Output:
Yes

System Output:
NextTurn();

5.2.4 Move to a pipe

Scenario:
The player takes action to move on to a pipe.

User Input:
1

System Output:
“Where do you want to move?
Enter 1 for pipe
Enter 2 for pump”
Enter 3 for cistern”

User Input:
 1

System Output:
 “There are currently 5 pipes.

Which pipe do you want to move to? (Enter a number from 1 to 5)”

User Input:
3

Note that this is an example, the user can choose any number from 1 to 5, e.g. 3

Condition Check:
The controller checks if the pipe is unoccupied and is available for a player to stand on.
pipe[x].Standable()==True

System Output:
Is Pipe [x] unoccupied?

User Input:
yes

System Output:
Player.move();
“You have moved to pipe 3”

5.2.5 Insert End of Pipe

Scenario: Plumber intends to insert the end of pipe to an element

User Input: 7

Condition Check: “Check if the player is at the end of the pipe”

System output: “Are you at the end of the pipe?”

User Input: “Yes”

System Output: InsertPipeEnd(e: EndOfPipe, e2: Element)

“If the player was not at the end of the pipe”

System Output: “Please Move to the end of the pipe to insert”

5.2.6 Pick up End of Pipe

Scenario: plumber picking up end of pipe

User Input: 6

Condition Check: “Check if the player position is at the end of the pipe”

System output: “Are you at the end of the pipe?”

User input: “Yes”

System Output: GetEnd(e: EndofPipe)
"The plumber has picked up an end of pipe. They can now insert it."

5.2.7 Fix Pipe

Scenario: Plumber navigates to a pipe to fix it
“Check if the pipe is occupied and if the pipe is broken”

System Output:”Are you on the Pipe?”

User Input: yes

System Output:”Is the pipe punctured?”

User Input: yes

Condition check:
“If the pipe is not occupied and it is broken: ”

```
isOccupied()== False
Pipe.Works == False
```

System output: “Pipe is now repaired and working.”

```
Plumber.FixPipe(Pipe)
Pipe.Works = True
```

5.2.8 Insert a pump into a pipe

Scenario: A plumber inserts a pump in the pipe system

Condition Check: the system checks if the user is on the pipe

System Output:
“Are you on the pipe you want to insert a pump on?”

User input:
“Yes”

Alt: System Output:
“Please move to the pipe you want to insert the pump on”

System Output: InsertPump(p1:Pump, p2:Pipe)
Pipe.connectToElement(Pump);

“Pump is successfully inserted into the pipe system.”
Alt:
”Please move to the pipe you want to insert the pump on”.

5.2.9 Pick up a Pump

Scenario: Plumber intends to pick up a pump.

Condition Check: “The system checks if the player is on a pump.”

System output: “Are you on the pump you want to pick up?”

User input: “Yes”

“The system allows the plumber to pick up the pump”

System Output: GetPump(p: Pump)

Alt: “ First go to a cistern that has a pump available for pick up”

System Output: “Please move to the pump in order to pick it up”

5.2.10 Fix a broken Pump

Scenario:
A plumber chooses to fix a broken pump.

User Input: 4

System Output: “Are you on the Pump?”

User Input: Yes

System Output: “Is the pump Working?”

User Input: No

Condition check:
Check the working state of the pump and if it's correctly occupied by the plumber.
Pump.IsWorking==False
PumpIsOccupied==True

System Output:
FixPump(Pump);
Pump.Works=True;
Pump is now repaired and working

5.2.11 Puncture a Pipe

Scenario: The saboteur navigates to a pipe location in the game to attempt to puncture a pipe.

User Input: 4

Note that 4 corresponds to Puncture when it is the Saboteur's turn to take action.

System Output: "Is the Pipe working? Enter 1 if yes, enter anything else if not."

User Input: 1

System Output: "Are you standing on the pipe? Enter 1 if yes, enter anything else if not."

User Input: 1

System output: Pipe.works = False
"The pipe has been punctured"
Condition Check: If water is currently flowing through the pipe:
System Output:
"decrementWater()"
"incrementLeakage()"

5.2.12 End Game and Determine Winner

Scenario: when the time is up the system ends the game, and concludes the results.

"System checks if the game timer has expired"

System output:

GetTime()

Condition check:
“If the time is up”

System output:
 EndGame()
 DetermineWinner()

Condition check:
DetermineWinner()==Plumbers

System output:
“Congratulations Team Plumbers, you win!”

Alt Condition check:
DetermineWinner()==Saboteurs

System output:
“Congratulations Team Saboteurs, you win!”

EndGame() calls DetermineWinner() method and DetermineWinner() internally calls: CalculateLeakedWater() to compute the total leaked water. CalculateCollectedWater() to compute the total collected water

---

0.1 Starting the Game

Scenario: Start Game

The start of the game happens after initializing the game. This phase marks the transition from game setup to active gameplay.

System Output: "Starting the game..."
Controller calls StartGame() method.
setTime(double)

System Output: "Game timer started."

System Output: "Players are placed on the map."

System Output: "Water flow has started."

System Output: "The game has started!"

The loop begins:

For each player in Team Plumbers:
  System Output: "Player [Player Name], it's your turn."
  System Output: "What action would you like to perform?"
  System Output: "[List of available actions for Plumbers]"

For each player in Team Saboteurs:
  System Output: "Player [Player Name], it's your turn."
  System Output: "What action would you like to perform?"
  System Output: "[List of available actions for Saboteurs]"

Note that the list of available actions for plumbers and saboteurs is clarified in 0.3.

0.2 Initialising the game

Initiating the game sequence is a process triggered by the system “controller” when the user launches the game. Once the game is launched and the player initiates gameplay, the Controller automatically begins setting up the game environment and preparing the elements for gameplay.

System Output: "How many players will participate? Please enter a number (4 or 6):"
User Input: The user enters the number of players (e.g. 4).

System Output: "You've selected [number] players."

Asks all the players
System Output: “Enter the name of the player [i]:”

System Output: "Select the team for [Player Name]: Enter '1' for Plumbers or '2' for Saboteurs and press Enter:"
Condition Check:
  If the selected team is not full:
    User input: The user joins the team they selected (either 1 or 2)
  Else:
    System Output: "[Player Name], You must join the [other team]."
    Assign the player to the team with fewer members.

System Output: "Initializing the game..."

System Output: "Adding pumps to the game..."
    Controller calls AddPump() method.
    Check: Confirm pumps are added
    System Output: "Pumps added successfully."

System Output: "Adding pipes to the game..."
    Controller calls AddPipe() method.
    Check: Confirm pipes are added
    System Output: "Pipes added successfully."

System Output: "Adding cisterns to the game..."
    Controller calls AddCistern() method.
    Check: Confirm cisterns are added
    System Output: "Cisterns added successfully."

System Output: "Adding springs to the game..."
    Controller calls AddSpring() method.
    Check: Confirm springs are added
    System Output: "Springs added successfully."

System Output: "Game initialization completed."

System Output: Controller calls startGame() method.
"Starting the game..."

0.3 Take Turn

Once the game is initialized and started, players will have the opportunity to choose their respective teams. After successfully selecting a team, each player will be able to perform unique actions based on their team affiliation.

For Plumbers:

System Output: "What action would you like to perform?"

System Output: Available actions for Plumbers:

1. Move to an element
2. GetPump
3. InsertPump
4. FixPump
5. FixPipe
6. GetEnd
7. InsertPipeEnd
8. ChangeInputPipe
9. ChangeOutputPipe
10. End Game

For saboteurs:

System Output: "What action would you like to perform?"

System Output: Available actions for Saboteurs:

1. Move to an element
2. ChangeInputPipe
3. ChangeOutputPipe
4. Puncture
5. End Game
