국민대학교

소프트웨어 프로젝트 II

Tic Tac Toe

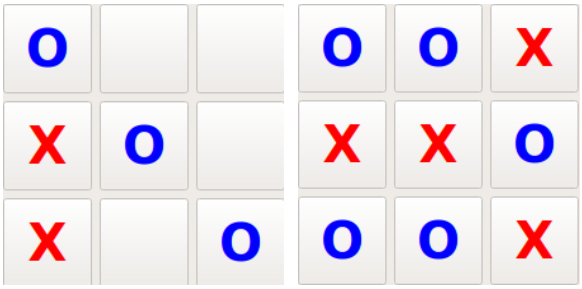
김주환, 백지수

2018-12-11

목차

I. 서론

[Tic Tac Toe] 게임은 3 × 3 판에 두 플레이어가 O 와 X 를 번갈아가며 써서 가로, 세로, 혹은 대각선 상에 같은 글자 세 개를 놓으면 이기는 게임이다. 간단히 말해, 작은 판에서 하는 오목이다. 예를 들어 아래의 [그림 1]의 왼쪽은 O 를 쓴 플레이어가 승리한 상황이다. 오른쪽은 O 와 X 가 모두 가로, 세로, 혹은 대각선 상에 같은 글자 세 개를 놓지 못했으므로 비긴 상황이다.



[그림 1] Tic Tac Toe 결과 예시

본 문서에서는 [Tic Tac Toe] 게임을 Python 과 PyQt5 를 이용해 GUI(Graphic User Interface) 기반 게임을 구현한 결과와 중간 산출물을 다룬다. 사용자와 인공지능이 게임을 진행하도록 구현되어 있으며, 인공지능은 3 단계의 난이도로 구현되어 있다.

II. 사항 명세서(SRS)

1. 기능적 요구사항

본 게임의 핵심적은 요소는 게임을 운용하는 클래스와 사용자의 입력에 맞게 버튼을 클릭하는 AI의 구현이다. 따라서 본 항에서는 게임을 운용하는 클래스의 기능적 요구사항과 AI의 기능적 요구사항을 다룬다. '게임을 운용하는 클래스'를 간단히 '게임'으로 나타내어 요구사항을 정리한 것이 [표 1]과 같다.

게임	● 게임이 종료되면 결과를 출력한다. ● 게임이 끝난 후에는 사용자 입력을 받아들이지 않아야 한다. ● 이미 선택된 입력은 다시 선택할 수 없어야 한다.
AI	● AI의 난이도를 쉽게 추가할 수 있어야 한다. ● 사용자가 무작위로 버튼을 선택할 때, 난이도가 증가할수록 AI의 승률이 높아져야 한다.

[표 1] 게임의 기능적 요구사항

2. 사용자 인터페이스 요구사항

게임의 사용자 인터페이스를 구성하는 요소는 현재 결과를 출력하는 디스플레이, 게임을 진행하는 게임 판, AI의 난이도를 선택하는 AI 난이도 선택 창, 게임을 새로 시작하는 창으로 구성되어 있다. 각 항목별로 요구사항을 정리한 것이 [표 2]와 같다.

디스플레이	● 표시되어 있는 글자를 사용자가 수정할 수 없어야 한다. ● 현재 몇 번째 턴인지 확인할 수 있어야 한다. ● 게임이 종료된 경우, 게임의 결과를 확인할 수 있어야 한다.
게임 판	● 게임의 진행 상태를 버튼에 표시해야 한다.
AI 난이도 선택 창	● 현재 AI의 난이도를 표시해야 한다. ● 버튼을 클릭할 시, 선택할 수 있는 난이도의 목록을 표시해야 한다.

[표 2] 사용자 인터페이스 요구사항

3. 비기능적 요구사항

이 소프트웨어의 구현은 Python 언어를 사용하고, PyQt5 패키지를 사용하여 GUI를 구현한다.

III. 소프트웨어 구조 설계서(ADS)

1. 소프트웨어 구조 설계(Architecture Design)

게임의 구현 시 사용할 모듈은 TicTacToe.py 와 ai.py 이다. 각 모듈들 사이의 상호 작용과 모듈에 속해있는 클래스를 정리한 것이 [표 3]이다.

모듈	클래스	역할
TicTacToe.py	Button	게임판의 버튼의 크기와 폰트를 지정해줌
	TicTacToe	사용자 인터페이스의 대부분의 위젯을 포함하는 UI component 게임 로직 구현
ai.py	AI	AI 로직 구현

[표 3] 모듈간의 상호작용

2. 클래스 인터페이스 설계

각 클래스의 메서드를 간략히 정리하였다. Button 클래스는 QToolButton 의 매서드를 override 한 것만 존재하므로 생략하였다.

클래스	메서드	입력인자	출력인자	기능
TicTacToe	gameButtonClicked	-	-	게임 판의 버튼들에 대한 콜백 사용자의 입력을 받아들여 처리
	gameStart	-		"New Game" 버튼에 대한 콜백 출력과 게임 판을 초기화
	setCurrent	-		게임판의 상태를 갱신
	gameEnd	-	winner	게임이 끝났는지 확인하고, 끝났다면 승자가 누구인지 출력
AI	guess	current, turn	row, col	AI 가 선택한 좌표를 반환
	getRandomCoordinate	current	row, col	아직 선택하지 않은 버튼 중 임의로 하나를 선택
	getHVDnum	current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum	-	현재 게임판에서 가로로 적힌 O, X 각각의 개수, 세로로 적힌 개수, 대각선으로 적힌 개수를 계산
	getDoubleCoordinate	current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum	row, col	Fork(다음 턴에 오목의 33 과 같은 수가 되는 경우)가 발생하는지 판단하여 이를 막기 위한 좌표 반환

[표 4] 클래스 인터페이스

3. 전체 구조

이상의 결과를 정리하여 하나의 그림으로 정리한 것이 [그림 2]이다.

TicTacToe 클래스가 게임의 운용 및 사용자와의 인터페이스를 담당하고, AI 는 사용자의 입력에 맞게 적절한 좌표를 TicTacToe 클래스에 전달한다.

[그림 2] 소프트웨어 구조 설계서서

IV. 소프트웨어 상세 설계서 (DDS)

본 절에서는 모듈별 구현 방식을 상세히 기술한다. 구현 방식을 자료 구조와 알고리즘을 중심으로 설명하되, GUI 구현은 일반적인 구현 방식(수업시간 중 다루었던 방식)을 이용하므로 이에 대한 설명은 생략하였다.

1. TicTacToe.py

Attributes (propertis)	turn	게임 진행 횟수를 가지고 있는 정수
	current	현재 게임 판의 상태를 가지고 있는 3 × 3 이차원 리스트 사용되지 않은 버튼이면 -1, 사용자가 누른 버튼이면 0, AI 가 누른 버튼이면 1 을 저장
Methods	gameButtonClicked	사용자가 버튼을 눌렀을 때 게임을 진행시킨다.
	gameStart	새 게임의 시작 attributes 를 초기화 하고, 버튼을 모두 누를 수 있는 상태로 바꾼다.
	setCurrent	current 를 현재 게임 판의 상태에 맞게 갱신한다.
	gameEnd	게임이 종료되었는지 확인하고, 종료되었다면 승자는 누구인지 리턴한다.

[표 5] TicTacToe 모듈 요약

1) gameButtonClicked

버튼이 눌렸을 때 게임을 진행시키는 메서드이다. 버튼이 눌렸다면 사용자가 누른 것을 표시할 수 있도록 버튼의 text 를 'O'로 바꾸고 버튼을 더 이상 누를 수 없게 상태를 바꾼다. turn 을 하나 증가시킨다. setCurrent 매서드를 호출해 current 를 갱신하고, AI 클래스의 guess 메서드를 호출해 AI 플레이를 진행한다.

2) gameStart

새 게임을 시작하는 메서드이다. attributes 를 초기화한다.

3) setCurrent

current 의 상태를 갱신하는 역할이다. 이중 반복문을 이용하여 각 원소를 순환한다. 버튼에 쓰인 글자가 'O'이면 current 의 값을 0 으로 'X'이면 current 의 값은 1 로 아직 선택되지 않았다면 -1 로 바꾼다.

4) gameEnd

게임이 끝났는지 확인하는 역할이다. 게임이 끝났다는 말은 가로, 세로, 대각선 중 같은 글자가 3 번 반복되는 것이 존재한다는 의미이므로 각 행별, 열별, 대각선별 글자의 수를 구한 뒤, 글자의 수가 3 인 경우 게임이 종료되었음을 알리고 어떤 글자가 3 번 반복되었는지 반환한다.

이를 구현하기 위해 행별 문자의 개수를 세는 3 × 2 배열 horizonRepeatNum, 열별 문자의 개수를 세는 3 × 2 배열 verticalRepeatNum, 대각선별 반복되는 문자의 개수를 세는 2 × 2 배열 diagonal-RepeatNum 을 사용하였다. 이때 각 배열의 열은 문자의 형태를 나타낸다. 예컨대 horizonRepeat-Num 의 0 행 0 열은 0 행의 'O' 문자의 개수를 나타내고, 0 행 1 열은 0 행의 'X' 문자 개수를 나타낸다.

2. ai.py

Attributes (propertis)	Level	AI의 난이도를 저장하는 문자형 변수
Methods	guess	AI가 선택한 좌표를 반환한다.
	getRandomCoordinate	이전에 선택하지 않은 임의의 좌표를 반환한다.
	getHVDnum	행, 열, 대각선별 문자의 개수를 구한다.
	getDoubleCoordinate	fork(오목의 33과 같은 경우)가 다음 턴에 발생하는지 확인하고, 발생한다면 이를 막기 위한 좌표를 반환한다.

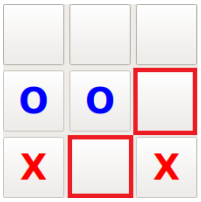
[표 6] ai.py 모듈 요약

1) guess

current와 turn을 입력으로 받아 좌표를 선택해 반환하는 메서드이다. 난이도를 easy, normal, hard로 나누어 구현하였다.

Easy의 경우 임의의 좌표를 선택하여 반환한다.

Normal은 두 문자가 반복되는 곳을 우선 선택하고, 이러한 곳이 없다면 다시 임의의 좌표를 선택하여 반환한다. 즉, [그림 3]에서는 빨간 박스로 표시한 두 부분 중 하나를 선택하도록 구현하였다.



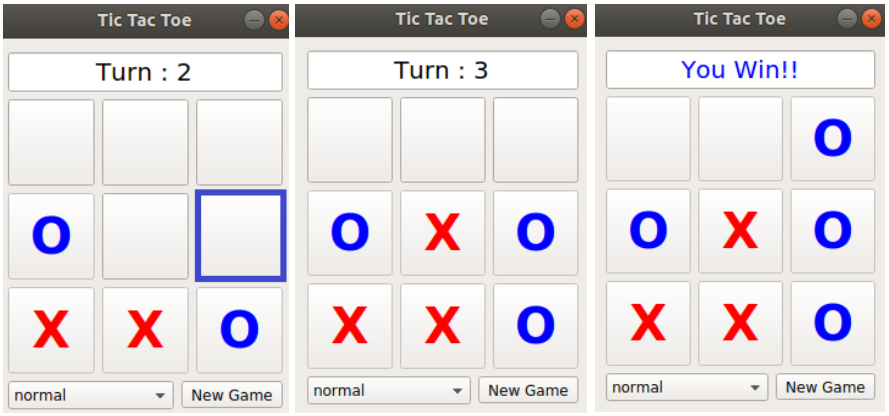
[그림 3] TicTacToe normal 예시

Hard는 처음 시행에서 사용자가 중앙을 선택하였다면 corner(예를 들어(0, 0)) 중 하나, edge((1, 0), (0, 1), (2, 1), (1, 2))을 선택하였다면 해당 위치와 가까운 corner, corner를 선택하였다면 중앙을 선택하도록 구현하였다. 이와 같이 선택하는 이유는 각 경우가 최상의 수이기 때문이다.[1]

두 번째 턴 이후부터는, normal과 마찬가지로 두 문자가 반복되는 곳이 있는지 먼저 확인하고, 이러한 곳이 없다면 다음 턴에 fork가 발생하는지 확인한다. fork도 발생하지 않는다면 임의로 좌표를 선택한다.

[1] Tic-tac-toe, wikipeida, https://en.wikipedia.org/wiki/Tic-tac-toe, 2018.12.07 자료 확인

fork 에 대해 더 자세히 설명하면 다음과 같다. [그림 4]의 첫 번째는 다음 턴에 fork 가 발생할 수 있는 경우이다. 'O'를 쓰는 사용자가 다음 턴에 파란색 박스를 선택하면 오목의 33 과 같은 경우가 발생하게 된다. fork 가 이미 발생하였다면, 중앙의 그림처럼 방어하여도 4 번째 turn 에서 지게 된다. 따라서 다음 턴에 fork 가 발생하는지 확인하고, AI 가 왼쪽 그림의 파란색 지점을 먼저 선택할 수 있도록 해야 한다.



[그림 4] fork 설명

2) getRandomCoordinate

좌표를 임의로 선택한다. 좌표의 선택은 random 모듈의 randint 함수를 이용하였다. 우선 임의로 좌표를 하나 선택한 뒤, current 리스트를 이용해 이미 선택된 좌표인지 확인한다. 만약 이미 선택된 좌표라면 while 문을 이용해 좌표를 다시 선택할 수 있도록 한다.

3) getHVDnum

TicTacToe class 의 gameEnd 와 유사한 논리를 사용한다. current 를 입력으로 받아 horizonRepeat-Num, verticalRepeatNum, diagonalRepeatNum 을 설정한다.

이중 반복문을 통해 리스트의 각 원소를 탐색한다. current 의 원소가 -1 이 아니면(즉, 이전에 선택된 버튼이라면) 리스트의 값 증가시킨다.

4) getDoubleCoordinate

fork(오목의 33 과 같은 경우)가 다음 턴에 발생하는지 확인한다. fork 가 발생하려면, 'O'만 1 번 쓰인 행과 열의 조합, 행과 대각선의 조합, 열과 대각선의 조합이 존재해야 한다. 따라서 getHVD-num 으로 구한 리스트를 활용해 위의 경우가 존재하는지 판단하고, 존재한다면 행과 열, 행과 대각선, 열과 대각선이 중첩되는 지점이 사용자가 다음 턴에 fork 를 만들기 위해 선택해야 할 지점이므로 AI 가 이 지점을 선택할 수 있도록 한다.

V. 소스코드

1. TicTacToe.py

```
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtWidgets import QLineEdit, QToolButton, QComboBox
from PyQt5.QtWidgets import QSizePolicy
from PyQt5.QtWidgets import QLayout, QGridLayout

from time import sleep
from ai import AI

class Button(QToolButton):



    def __init__(self, callback):

        super().__init__()
        self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Preferred)
        self.clicked.connect(callback)

        font = self.font()
        font.setFamily("Curier New")
        font.setBold(True)
        font.setPointSize(30)
        self.setFont(font)



    def sizeHint(self):

        size = super(Button, self).sizeHint()
        size.setHeight(70)
        size.setWidth(size.height())
        return size

class TicTacToe(QWidget):



    def __init__(self, parent=None):

        self.turn = 0
        super().__init__(parent)
        self.current = [[-1] * 3 for i in range(3)]

        # Display Window
        self.display = QLineEdit("Your Turn!")
        self.display.setReadOnly(True)
        self.display.setAlignment(Qt.AlignHCenter)
        font = self.display.font()
        font.setPointSize(15)
        font.setFamily("Curier New")
        self.display.setFont(font)

        # game grid create
        gameLayout = QGridLayout()
        self.Buttons = [[] * 3 for i in range(3)]
        for row in range(3):
            for col in range(3):
                self.Buttons[row].append(Button(self.gameButtonClicked))
                gameLayout.addWidget(self.Buttons[row][col], row, col)
```

```
        # AI level comboBox
        self.qb = QComboBox()
        self.qb.addItem("easy")
        self.qb.addItem("normal")
        self.qb.addItem("hard")
        self.qb.currentTextChanged.connect(self.gameStart)
        self.ai = AI(self.qb.currentText())

        # Button for a new game
        self.newGameButton = QToolButton()
        self.newGameButton.setText("New Game")
        self.newGameButton.clicked.connect(self.gameStart)

        # Layout
        mainLayout = QGridLayout()
        mainLayout.setSizeConstraint(QLayout.SetFixedSize)
        mainLayout.addWidget(self.display, 0, 0, 1, 2)
        mainLayout.addLayout(gameLayout, 1, 0, 1, 2)
        mainLayout.addWidget(self.qb, 2, 0, 1, 1)
        mainLayout.addWidget(self.newGameButton, 2, 1, 1, 1)

        self.setLayout(mainLayout)

        self.setWindowTitle("Tic Tac Toe")

        self.gameStart()



def gameButtonClicked(self):
        # User Turn
        clickedButton = self.sender()
        clickedButton.setText("O")
        clickedButton.setStyleSheet('QToolButton{color: blue;}')
        clickedButton.setEnabled(False)
        self.setCurrent()

        if self.gameEnd() == 0:
            self.display.setText("You Win!!")
            self.display.setStyleSheet('QLineEdit{color: blue;}')
            for r in range(3):
                for c in range(3):
                    self.Buttons[r][c].setEnabled(False)
            return

        self.turn += 1
        self.display.setText("Turn : " + str(self.turn))
        if self.turn == 5:
            self.display.setText("Draw!")
            return

        # AI Turn
        self.setCurrent()
        row, col = self.ai.guess(self.current, self.turn)
        self.Buttons[row][col].setText("X")
        self.Buttons[row][col].setStyleSheet('QToolButton{color: red;}')
        self.Buttons[row][col].setEnabled(False)
        self.setCurrent()

        if self.gameEnd() == 1:
            self.display.setText("AI Win!!")
            self.display.setStyleSheet('QLineEdit{color: red;}')
            for r in range(3):
                for c in range(3):
                    self.Buttons[r][c].setEnabled(False)
            return
```

```
def gameStart(self):
        self.display.setText("Your Turn!")
        self.display.setStyleSheet('QLineEdit{color: black;}')
        self.ai = AI(self.qb.currentText())
        self.setCurrent()
        self.turn = 0

        for row in range(3):
            for col in range(3):
                self.Buttons[row][col].setText("")
                self.Buttons[row][col].setEnabled(True)



    def setCurrent(self):
        for row in range(3):
            for col in range(3):
                text = self.Buttons[row][col].text()
                if text == "O":
                    self.current[row][col] = 0
                elif text == "X":
                    self.current[row][col] = 1
                else:
                    self.current[row][col] = -1



    def gameEnd(self):
        horizonRepeatNum  = [[0, 0] for i in range(3)]
        verticalRepeatNum = [[0, 0] for i in range(3)]
        diagonalRepeatNum = [[0, 0] for i in range(2)]

        # get RepeatNum
        for row in range(3):
            for col in range(3):
                currentValue = self.current[row][col]
                if currentValue == -1:
                    continue
                horizonRepeatNum[row][currentValue]    += 1
                verticalRepeatNum[col][currentValue]   += 1

                if row == col:
                    diagonalRepeatNum[0][currentValue] += 1
                if row == 2 - col:
                    diagonalRepeatNum[1][currentValue] += 1



        for i in range(3):
            if horizonRepeatNum[i][0] == 3 or verticalRepeatNum[i][0] == 3 or diagonalRepeatNum[0][0] == 3 or
diagonalRepeatNum[1][0] == 3:
                return 0

            elif horizonRepeatNum[i][1] == 3 or verticalRepeatNum[i][1] == 3 or diagonalRepeatNum[0][1] == 3 or
diagonalRepeatNum[1][1] == 3:
                return 1

        return -1



if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)
    game = TicTacToe()
    game.show()
    sys.exit(app.exec_())
```

2. ai.py

```
import random

class AI:
    def __init__(self, level):
        self.level = level

    def guess(self, current, turn):
        if self.level == "easy":
            return self.getRandomCoordinate(current)

        elif self.level == "normal":
            horizonRepeatNum  = [[0, 0] for i in range(3)]
            verticalRepeatNum = [[0, 0] for i in range(3)]
            diagonalRepeatNum = [[0, 0] for i in range(2)]
            self.getHVDnum(current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum)

            # if horizon repeat num is 2
            row, col = self.getDobuleCoordinate(current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum)
            if row != -1:
                return row, col

            # if don't exist repeat num 2
            return self.getRandomCoordinate(current)
```

```
elif self.level == "hard":
        if turn == 1:
            # center opening
            if current[1][1] == 0:
                return 0, 0

            # edge opening
            elif current[1][0] == 0 or current[0][1] == 0:
                return 0, 0
            elif current[2][1] == 0 or current[1][2] == 0:
                return 2, 2

            # corner opening
            else:
                return 1, 1

        horizonRepeatNum  = [[0, 0] for i in range(3)]
        verticalRepeatNum = [[0, 0] for i in range(3)]
        diagonalRepeatNum = [[0, 0] for i in range(2)]
        self.getHVDnum(current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum)

        # if horizon repeat num is 2
        row, col = self.getDobuleCoordinate(current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum)
        if row != -1:
            return row, col

        # if exist fork
        for hor in range(3):
            for ver in range(3):
                if (horizonRepeatNum[hor][0] == 1 and horizonRepeatNum[hor][1] == 0) and (verticalRepeatNum[ver][0] == 1 and
verticalRepeatNum[ver][1] == 0) and current[hor][ver] == -1:
                    return hor, ver

        if diagonalRepeatNum[0][0] == 1 and diagonalRepeatNum[0][1] == 0:
            for hor in range(3):
                if horizonRepeatNum[hor][0] == 1 and horizonRepeatNum[hor][1] == 0 and current[hor][hor] == -1:
                    return hor, hor

            for ver in range(3):
                if verticalRepeatNum[ver][0] == 1 and verticalRepeatNum[ver][1] == 0 and current[ver][ver] == -1:
                    return ver, ver

        if diagonalRepeatNum[1][0] == 1 and diagonalRepeatNum[1][1] == 0:
            for hor in range(3):
                if horizonRepeatNum[hor][0] == 1 and horizonRepeatNum[hor][1] == 0 and current[hor][2 - hor] == -1:
                    return hor, 2 - hor

            for ver in range(3):
                if verticalRepeatNum[ver][0] == 1 and verticalRepeatNum[ver][1] == 0 and current[ver][2 - ver] == -1:
                    return ver, 2 - ver

        # if don't exist repeat num 2
        return self.getRandomCoordinate(current)
```

```
def getRandomCoordinate(self, current):
        row = random.randint(0, 2)
        col = random.randint(0, 2)
        while current[row][col] != -1:
            row = random.randint(0, 2)
            col = random.randint(0, 2)

        return row, col


    def getHVDnum(self, current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum):
        # get RepeatNum
        for row in range(3):
            for col in range(3):
                currentValue = current[row][col]
                if currentValue == -1:
                    continue

                horizonRepeatNum[row][currentValue] += 1
                verticalRepeatNum[col][currentValue] += 1

                if row == col:
                    diagonalRepeatNum[0][currentValue] += 1
                if row == 2 - col:
                    diagonalRepeatNum[1][currentValue] += 1


    def getDobuleCoordinate(self, current, horizonRepeatNum, verticalRepeatNum, diagonalRepeatNum):
        for row in range(3):
            if horizonRepeatNum[row][0] == 2 or horizonRepeatNum[row][1] == 2:
                for col in range(3):
                    if current[row][col] == -1:
                        return row, col



        # if vertical repeat num is 2
        for col in range(3):
            if verticalRepeatNum[col][0] == 2 or verticalRepeatNum[col][1] == 2:
                for row in range(3):
                    if current[row][col] == -1:
                        return row, col

        # if digonal repeat num is 2
        if diagonalRepeatNum[0][0] == 2 or diagonalRepeatNum[0][1] == 2:
            for i in range(3):
                if current[i][i] == -1:
                    return i, i

        if diagonalRepeatNum[1][0] == 2 or diagonalRepeatNum[1][1] == 2:
            for i in range(3):
                if current[i][2 - i] == -1:
                    return i, 2 - i

        return -1, -1
```

VI. 통합 테스트 보고서

본 장에서는 클래스의 각 메서드를 테스트하고, AI 의 난이도에 맞게 알고리즘이 설정되었는지 점검한다.

1. 단위 테스트 (Unit Test)

게임이 GUI 기반으로 설계되었기 때문에, 게임을 실제로 동작시키며 오류를 발생시키지 않는지와 디버거를 이용해 값이 제대로 설정되는지 확인하였다. 이를 위해 동일한 기능의 수행을 10 번 반복하였다. [표 5]는 그 결과이다.

Test Case	UUT (Unit Under Test)	Description	Results
UT 1-1	AI.guess	AI 의 버튼 버튼 선택 테스트	Pass
UT 2-1	TicTacToe.gameButttonClicked	사용자 입력의 처리 기능 테스트	Pass
UT 2-2	TicTacToe.gameStart	새 게임의 시작 기능 테스트	Pass
UT 2-3	TicTacToe.setCurrent	현재 게임 판 상태 갱신 테스트	Pass
UT 2-4	TicTacToe.gameEnd	게임 종료 상태 확인 테스트	Pass

[표 7] 단위 테스트 결과

2. 통합 테스트 (Integration Test)

본 실험에서는 AI 의 난이도별로 5 번씩 실험을 반복하여 게임의 전반적인 실험에서 오류가 발생하지 않는지 확인한다. 또한 사용자의 입력을 랜덤하게 하였을 때, AI 의 난이도별로 승률을 측정하여 AI 의 난이도가 적절히 선택되었는지 확인한다. [표 6]의 result 는 오류가 발생하지 않았으면 Pass, 오류가 발생하였으면 Fail 로 표시하였고, winning rate 는 AI 의 승률을 나타낸다.

AI 난이도	Winning rate (%)	Results
Easy	60	Pass
Normal	100	Pass
Hard	100	Pass

[표 8] 통합 테스트 결과

사용자의 입력을 랜덤하게 할 경우 normal 이상의 난이도에 대해 측정하기 어렵기 때문에 실제 사용자의 입력으로 5 번씩 실험을 반복하여 난이도별 테스트를 수행하였다.

AI 난이도	Winning rate (%)	Results
Easy	20	Pass
Normal	50	Pass
Hard	100	Pass

[표 9] AI 테스트 결과