**2019년 2학기**

**프로그래밍과 문제해결**

**Assignment #3**

|  |  |
| --- | --- |
| 담당교수 | **윤은영** |
| 학번 | **20190084** |
| 학과 | **무은재학부** |
| 이름 | **권민재** |
| POVIS ID | **mzg00** |

|  |
| --- |
| **명예서약(Honor code)**  “나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.” |

**Problem: ‘Dots and boxes’ game**

**1. 문제의 개요**

본 프로그램의 개요는 아래와 같다.

* 본 프로그램에서는 4X4 크기의 ‘Dots and boxes’ 게임을 구현하고자 한다.
* 게임 판은 가로로 다섯 개, 세로로 다섯 개인 총 스물다섯개의 점으로 이루어져 있다.
* 총 플레이어의 수는 2명이며, 가로 또는 세로로 길이 1의 점을 그을 수 있다.
* 두 플레이이어의 턴이 번갈아가면서 진행되며, 플레이어가 선을 그어 닫힌 사각형을 만들면 1점을 득점한다. 득점할 경우 추가 턴을 얻는다.
* 본 프로그램은 플레이어와 플레이어, 플레이어와 인공지능, 인공지능과 인공지능 간의 대결을 지원하며, 인공지능의 난이도는 easy, normal, hard 총 3개가 구현되어 있다.

이 때 사용되는 구조 차트는 아래와 같이 표현될 수 있다.

스크린샷이(가) 표시된 사진

자동 생성된 설명

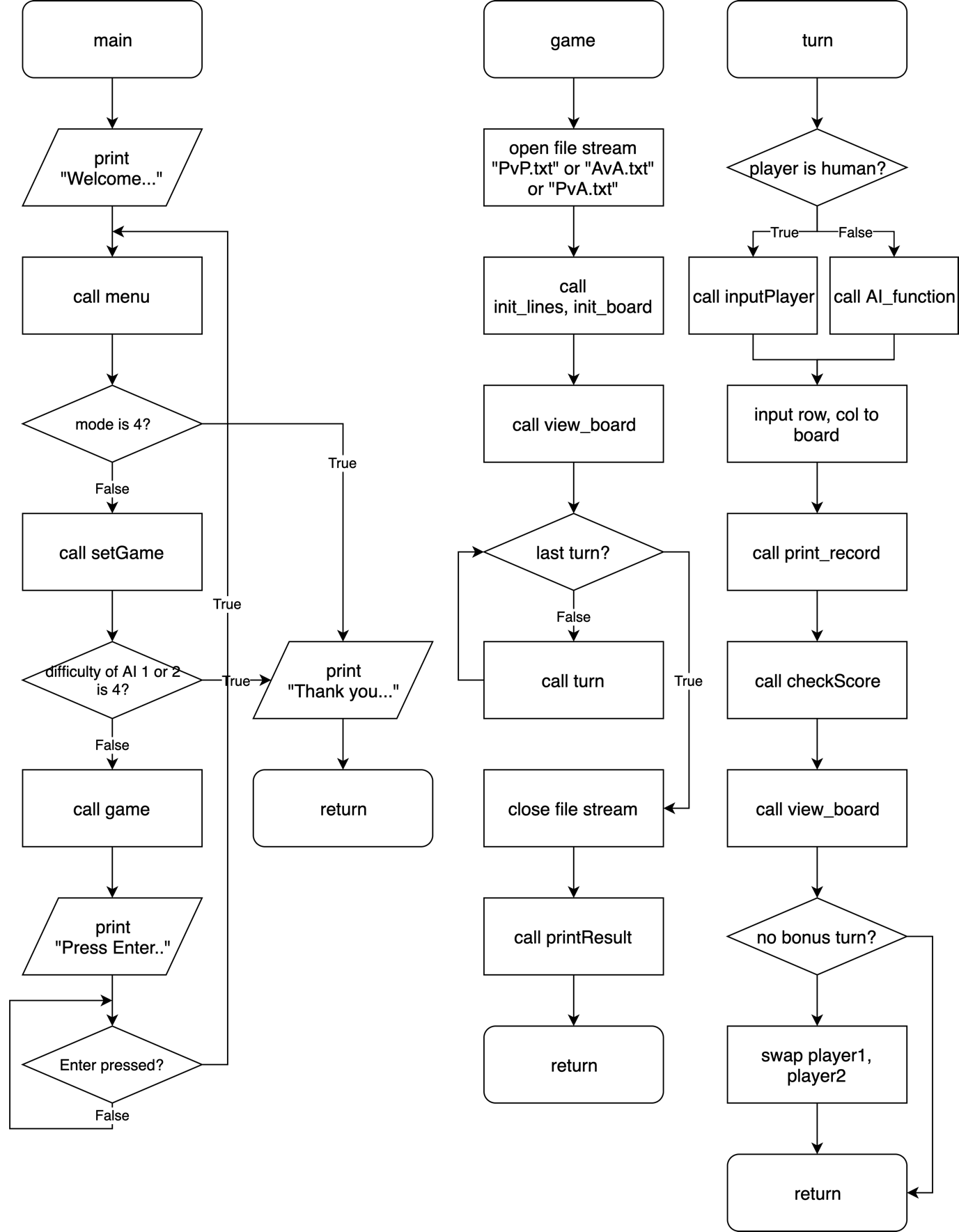
* **입력부:** 입력부는 해당 프로그램에서 사용자로부터의 입력을 담당한다. 첫째로, 사용자로부터 메뉴를 입력 받아 해당 게임이 플레이어와 플레이어, 플레이어와 인공지능, 인공지능과 인공지능 사이의 게임 중 어떤 게임인지 모드를 입력받으며, 그에 따라 인공지능의 난이도를 입력받는다. 둘째로, 사용자로부터 좌표를 입력받는다. 또한, 게임이 종료된 이후 엔터를 눌렀는지에 대한 입력도 담당한다.
* **처리부:** 처리부는 입력값에 맞게 프로그램을 진행한다. 사용자가 입력한 좌표에 맞게 게임 판에 선을 그으며, 그어진 선에 따라서 점수를 측정한다. 또한, 인공지능의 수를 처리한다. 인공지능의 난이도가 easy일 경우에는 무작위로 선을 선택하며, normal일 때는 바로 점수를 낼 수 있는 상황, 즉 선이 3개로 둘러싸인 경우에 남은 부분에 선을 그으며, hard일 때는 alpha-beta pruning이 적용된 minimax를 통해 수를 선택한다.
* **출력부:** 메인 메뉴와 인공지능 난이도 선택 메뉴의 출력, 게임 판과 같은 게임 진행 상황에 대한 출력을 담당한다.

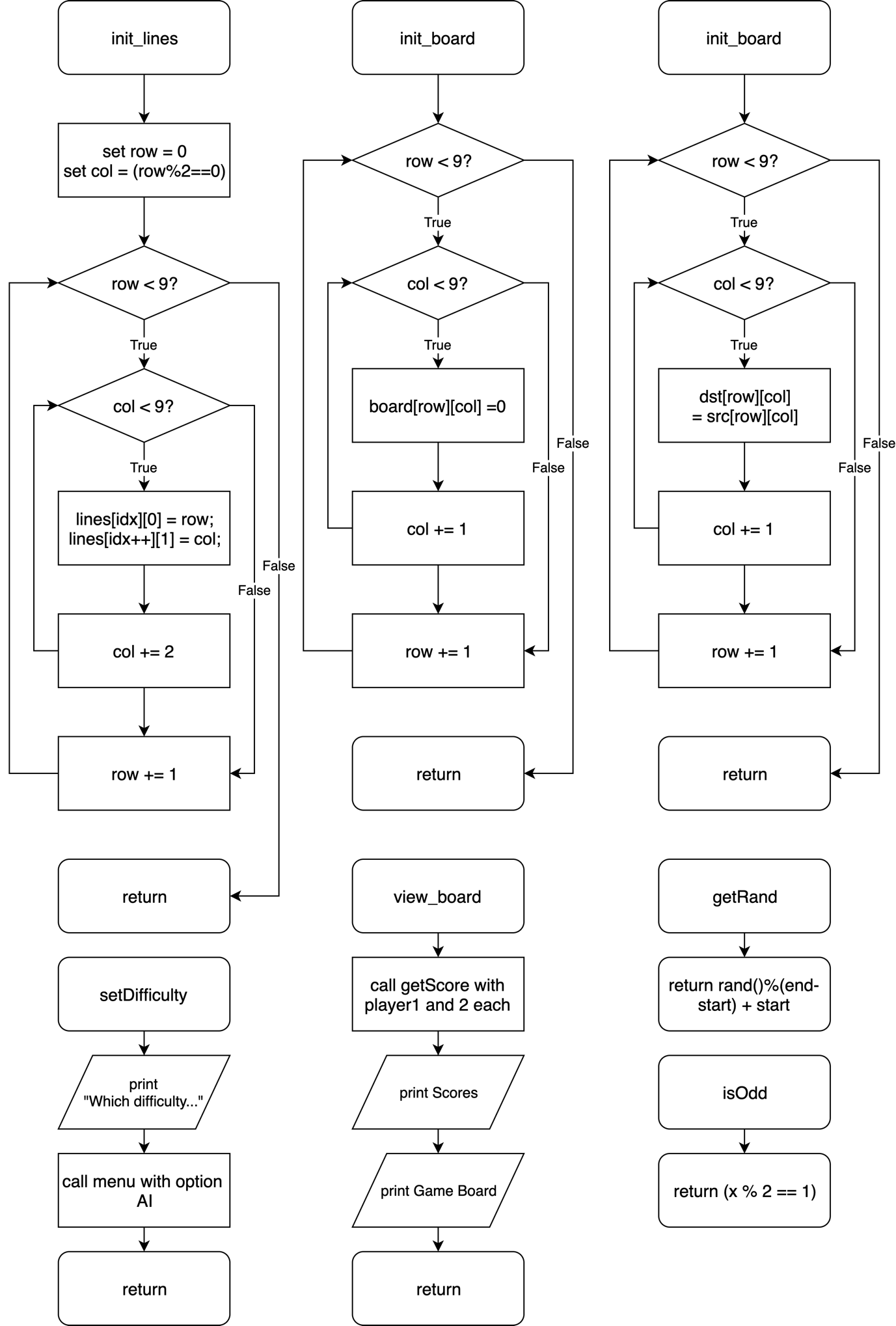
**2. 알고리즘**

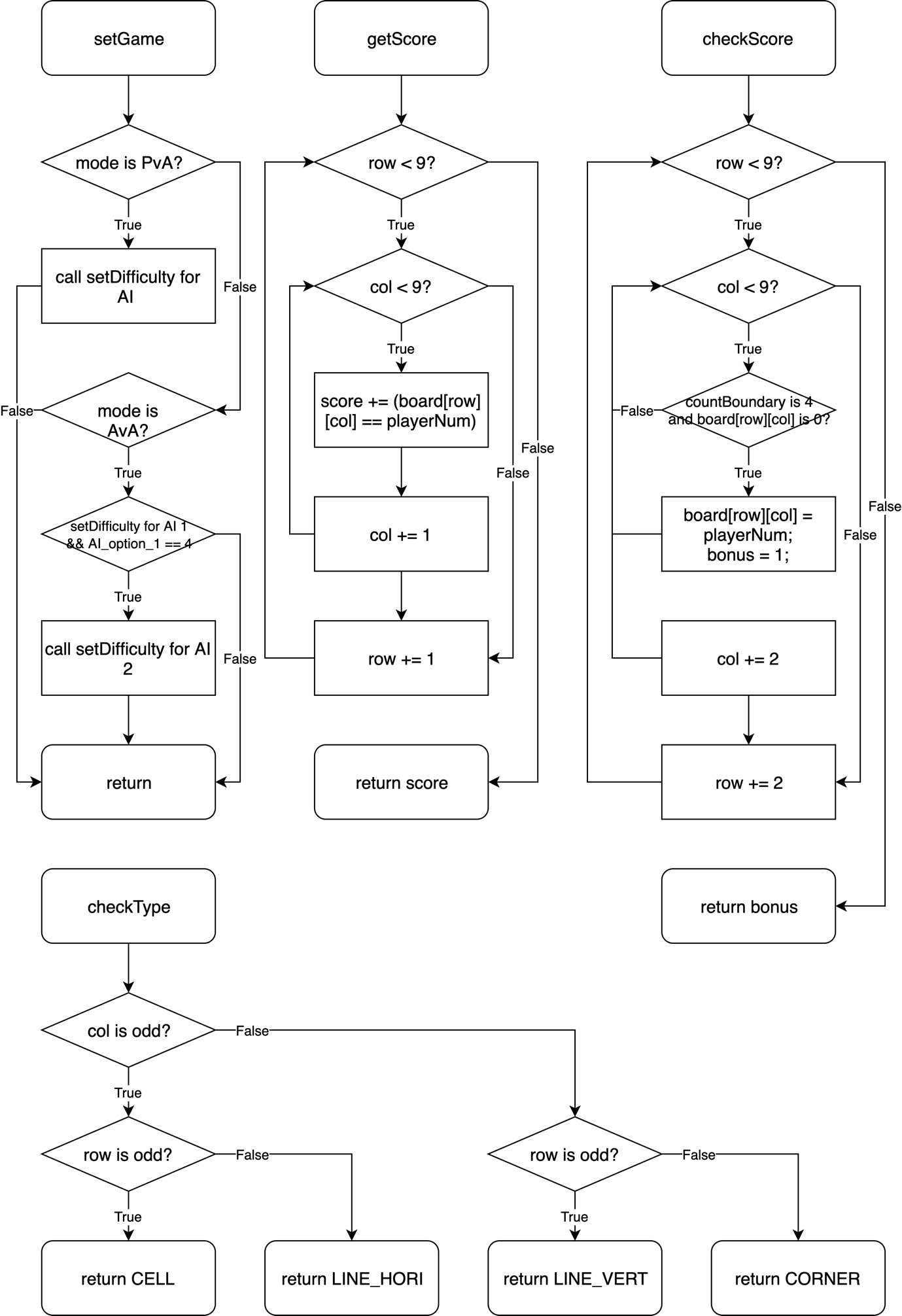
해당 프로그램의 의사 코드는 아래와 같다.

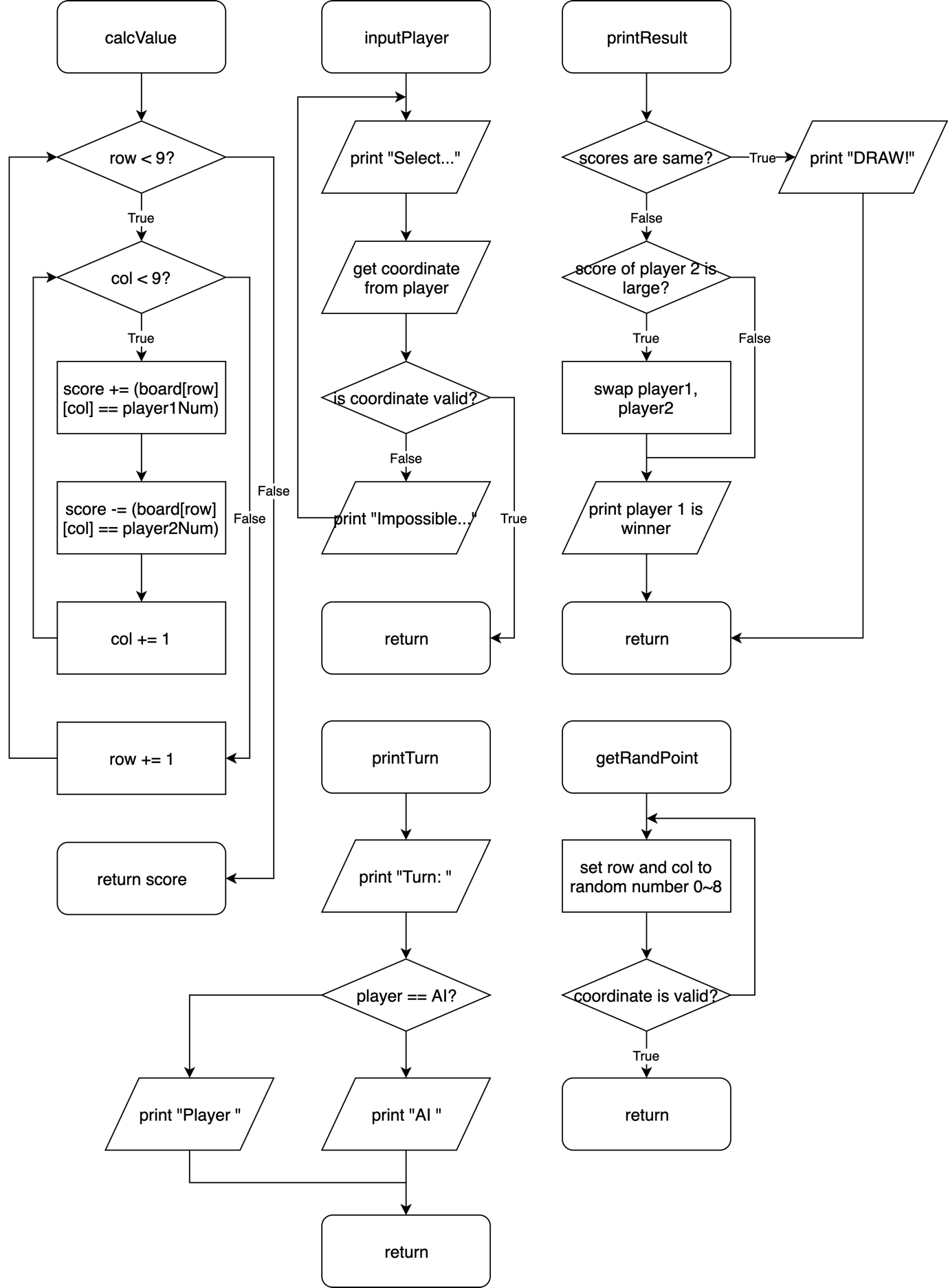
|  |  |
| --- | --- |
| Pseudo-code for ASSN3 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221 | // 프로그램에 필요한 변수들과 전처리 상수들은 미리 선언한 것으로 가정하자.  include essential headers <stdio.h>, <unistd.h>, <stdlib.h>, <time.h>  function *game*  open file “PvP.txt” or “AvA.txt” or “PvA.txt” with “w” mode.  init lines with calling *init\_lines*. // ‘lines’ is a array contain information about coordinate. For example, lines[0] == [0, 1], lines[1] == [0, 3] and so on.  init game board with calling *init\_board.*  print game board with calling *view\_board*.  call *turn* for 40 times. // 40 is max turn for 5x5 game.  close file stream.  print result with calling *printResult*.    function *turn*  if player is human, get input from human with calling *inputPlayer*.  else if, player is AI, get input from AI with calling *AI\_function*.  write the input coordinates to game board.  print to file with calling *print\_record*  check bonus turn with calling *checkScore*  print game board with calling *view\_board*.  if no bonus turn, change player1 and player2. with calling *swap*.  function *init\_lines*  set array ‘lines’. Array ‘lines’ matches numbers from 0 to 39 and coordinates of game board  function *init\_board*  set all elements of ‘board’ into 0.  function *view\_board*  check score of player1 and player2 by calling *getScore*.  print scores.  print “ “, “ --- “, “+”, “|”, “ O “, “ X “ by condition.  all of printing line consists of letter “=” is implemented by calling *printLine*  function *copy\_board*  copy game board source to destination.  function *setDifficulty*  print menu for setting AI difficulty by calling *menu*.  function *setGame*  set variable player1 and player2 with P or AI depend on its condition. P means Player.  if mode is PvA, set AI difficulty for AI.  else if mode is AvA, set AI difficulty for AI 1 and AI 2.  function *getScore*  get score of targeted player number with read game board.  function *checkScore*  if player get score, set that cell into that player’s number, and return bonus turn is 1.  function *checkType*  Return type of coordinates. Return value can be CELL or LINE\_HORI or LINE\_VERT or CORNER.  function *calcValue*  Return ((score of player1) – (score of player2)). It will be criterion for hard AI scoring nodes.  function *inputPlayer*  loop.  input coordinates from player.  if player’s input is valid, escape from this loop.  else, print why player’s input is not valid.  function *printResult*  if scores of two players are same, print DRAW  else, print winner properly.  function *printTurn*  print information of whose turn.  function *getRand*  return random number of input range.  function *getRandPoint*  loop until row and col is valid,  set row and col with calling *getRand*  function *AI\_function*  if difficulty of AI is easy, call *AI\_easy*.  else if, difficulty of AI is normal, call *AI\_normal*.  else if, difficulty of AI is hard, call *AI\_hard*.  function *AI\_easy*  set row and col with valid random coordinates with calling *getRandPoint*  function *AI\_normal*  loop in rows and cols of coordinates of boxes.  if box is surrounded with 3 lines, call *openBoundary* to set row and col with remaing coordinate.  if there is no box that surrounded with 3 lines, set row and col with calling *getRandPoint.*  function *AI\_hard*  if the current turn is less than 15, call *alpha\_beta\_minimax* with depth 5.  else if the current turn is less than 25, call *alpha\_beta\_minimax* with depth 6.  else, call *alpha\_beta\_minimax* with depth 7.  function *alpha\_beta\_minimax*  if depth is 0 and less, return prevScore.    if AI’s turn, set optimalChildValue to -1000.  else, set optimalChildValue to 1000.  for row and col in lines  copy game board to simulBoard with calling *copy\_board*  if AI’s turn, set simulBoard[row][col] for player1\_Num, and check bonus turn.  else, set simulBoard[row][col] for player2\_Num, and check bonus turn.  get expectedValue by calling *calcValue*    if there’s no valid line, push expected value and -1 to childList.  else,  get childValue from calling *alpha\_beta\_minimax.* (recursive)    if AI’s turn, optimalChildValue will become the larger of the two, optimalChildValue or childValue. And alpha value will become the larger of alpha or optimalChildValue.  else, optimalChildValue will become the smaller of the two, optimalChildValue or childValue. And beta value will become the smaller of two, beta or optimalChildValue.    push childValue and row&col into childList.  if alpha is bigger than beta, break.    if AI’s turn, set optimalValue into largest score in childList and set row and col with corresponding coordinate.  else, set optimalValue into smallest score in childList and set row and col with corresponding coordinate.  if set row and col is not valid, get random point with call *getRandPoint*.  return optimalValue.  function *print\_record*  print row and col into file.  function *printLine*  print "======================\n"  function *printMenu*  call printLine  print "1. Game: Player vs Player\n"  print "2. Game: Player vs AI\n"  print "3. Game: AI vs AI\n"  print "4. Quit\n"  call printLine  print "Select menu number: "  function *printMenu\_AI*  call printLine  print "1. Easy\n"  print "2. Normal\n"  print "3. Hard\n"  print "4. Quit\n"  call printLine  print "Select menu number: "  function *getMenu*  call printMenu  get input from user.  function *getMenu\_AI*  call printMenu\_AI  get input from user.  function *inputMenu*  if input is not one of value 1, 2, 3, 4, return 1.  if option is AI, set mode to difficulty. return 0.  function *menu*  loop until input is valid.  if option is AI, call getMenu\_AI.  else call getMenu  function *boundary*  set bound[0] is value of upper line.  set bound[1] is value of lower line.  set bound[2] is value of left line.  set bound[3] is value of right line.  function *countBoundary*  set bound with calling *boundary*.  for line in bound, count number of line.  return number of line.  function *openBoundary*  for row and col in lines near cell,  if line with row and col is valid, set row and col with it.  function *isOdd*  return true if input number is odd.  function *isValid*  return true if input row and col is line and not exceed index of board.  function *isLine*  return true if check type of row and col is LINE\_VERT of LINE\_HORI.  function *max*  return the large number of two  function *min*  return the small number of two  function *swap*  swap the given two numbers.  function *main*  print “Welcome to Dots and boxes!”  loop  get mode value by calling *menu*.  if mode value == 4, break.  set game parameters with calling *setGame*  if difficulty of AI 1 or AI 2 is 4, continue.  execute game with calling *game*  print “Press Enter key to return to the Main Menu.”  wait until press Enter key.  print “Thank you for playing Dots and boxes!” |

위의 의사 코드를 flowchart를 이용해서 나타내면 아래와 같다. (다음 페이지)









스크린샷이(가) 표시된 사진

자동 생성된 설명

측정기, 주차장이(가) 표시된 사진

자동 생성된 설명

실외, 전자기기, 검은색, 앉아있는이(가) 표시된 사진

자동 생성된 설명

**3. 프로그램 구조 및 설명**

**a) 메인 메뉴**

* 메인 메뉴는 기본적으로 무한 루프에 갇힌 구조이다.
* menu(&mode, P)를 호출하여 메인 메뉴를 출력하고, 입력 값을 mode에 저장한다.
* mode의 값이 4라면, 무한 루프에서 탈출한다.
* setGame(mode, &player1, &player2, &AI\_option\_1, &AI\_option\_2)를 호출하여 게임에 필요한 변수들을 세팅한다. setGame 안에서는 mode에 따라서 setDifficulty를 호출하여 AI 난이도를 선택하게 한다.
* 만약, AI\_option\_1이나 2에 담긴 값이 4라면, 난이도 선택을 나갔다는 의미이므로, continue를 이용해 메인 메뉴 루프의 제일 처음으로 보낸다.
* game(player1, player2, AI\_option\_1, AI\_option\_2)를 호출하여 세팅된 파라미터로 게임을 시작한다.
* 게임이 종료되면, 엔터 키를 눌러서 메인 메뉴로 갈 수 있다는 문장을 출력한다.
* while(getchar() != ‘\n’)을 이용해서 엔터 키가 눌릴 때까지 대기한다.
* 무한 루프를 탈출하였다면 “Thank you for playing Dots and Boxes!”를 출력할 수 있도록 했다.
* menu, setGame 등의 메뉴 선택을 할 수 있는 함수 안에는 입력을 검증하는 과정이 포함되어 있다.

**b) 게임 진행**

* 함수 game에서 게임 1번의 호흡을 담당한다.
* game에 게임 판에 대한 정보가 담기는 배열 board가 선언되며, 0~39의 숫자와 선들의 좌표가 1대 1 대응되어 있는 배열 lines가 선언된다.
* player1과 player2의 속성에 따라 “PvP.txt”, “AvA.txt”, “PvA.txt” 중 맞는 파일을 “w” 모드로 연다.
* lines를 init\_lines(lines)를 호출하여 초기화하고, board를 init\_board(board)를 호출해서 초기화한다.
* 게임이 시작할 때, view\_board를 호출하여 게임판을 우선 보여준다.
* 이 게임은 항상 40턴이 진행되어야 하므로, turn 함수를 40번 실행한다.
* 이때, turn 함수는 lines, board, 플레이어 1과 2의 속성과 플레이어 번호, 점수, AI 1과 2의 옵션, 현재 턴 수, 파일을 받으며, 현재 턴 수를 제외한 모든 매개변수는 포인터이다. 즉, 매개변수에 해당되는 game의 변수는 함수 turn에 의해 제어된다.
* 실행된 이후에는 printResult 함수를 호출하여 게임 결과를 보여주고 종료한다.

**c) 턴 진행**

* turn 함수에서는 player1을 현재 턴을 진행하고 있는 플레이어라고 생각하며, player2를 다른 플레이어라고 생각한다.
* 만약, player1이 사람이라면, inputPlayer를 호출하여 사람에게 입력받을 수 있도록 한다.
* inputPlayer에서는 사용자가 올바른 좌표를 입력할 때 까지 좌표를 입력 받고, turn에서의 좌표를 수정한다. (포인터를 쓴다는 의미이다.)
* 만약, player1이 인공지능이라면, 우선 PvA에서는 1초, AvA에서는 0.5초 만큼 usleep을 이용하여 쉬어준다. 이후, AI\_function을 호출하여 좌표 값을 입력받고, AI가 선택한 좌표를 출력한다.
* Hard 난이도를 포함한 인공지능의 자세한 매커니즘에 대해서는 **d)**에 따로 서술하였다.
* 입력 받은 좌표에 따라, board의 해당 좌표에 player1의 플레이어 숫자를 채워넣고, print\_record를 호출하여 이를 turn의 매개변수로 받은 파일에 기록한다.
* 이후, checkScore를 호출하여 추가 턴 여부를 확인한다. 함수 checkScore에서는 board의 box가 되는 좌표에 어떤 플레이어가 점수를 얻었는지 그 플레이어의 숫자를 채워 넣고, 점수를 얻었다면 1을 반환한다.
* 이후 view\_board를 호출하여 이번 턴의 결과를 보여준다.
* 만약, 추가 턴이 없다면, player1과 player2, player1\_Num과 player2\_Num, AI\_option1과 AI\_option2를 swap하여 차례를 바꿔준다. turn 함수에서 player1을 항상 현재 플레이어로 간주함을 생각하면 된다.

**d) 인공지능**

* AI\_function은 사용자가 지정한 난이도에 따라 AI\_easy, AI\_normal, AI\_hard를 호출해준다.
* AI\_easy에서는 getRandPoint를 호출하여 valid하면서 랜덤한 좌표 값을 turn의 좌표에 기입한다.
* getRandPoint는 valid한 좌표가 될 때 까지 계속해서 랜덤한 좌표를 찾는 함수이다.
* AI\_normal에서는 countBoundary를 호출하여 만약 한 박스 주변에 line이 3개가 있다면 openBoundary를 호출하여 남은 1자리의 좌표 값을 turn의 좌표에 기입해준다. 만약 3개인 곳이 없다면, getRandPoint를 호출하여 좌표 값을 기입한다.
* AI\_hard에서는 현재 진행된 턴 수에 따라 depth를 달리하여 alpha\_beta\_minimax를 호출한다. 현재 턴이 15 미만 이라면 depth는 5, 아니면서 현재 턴이 25 미만 이라면 depth는 6, 아니면 depth는 7이 된다.
* 초반에 깊은 depth를 탐색할 경우 시간이 과도하게 오래 걸리는 것을 방지하기 위함이다.
* 함수 alpha\_beta\_minimax는 **e)**에서 설명하고자 한다.

**e) Minimax with Alpha-Beta Pruning**

* 이 프로그램에서 Hard AI는 Alpha-beta Pruning이 적용된 minimax 알고리즘으로 구현되었다.
* 우선 Minimax란, 트리 가장 아래에서부터 나의 턴일 때는 child node 중 가장 점수가 큰 값을, 상대 턴일 때는 child node 중 가장 점수가 적은 값을 선택하여 최적의 수를 찾는 알고리즘이라고 말할 수 있다.
* 하지만, minimax만으로 수를 탐색하는데는 오랜 시간이 걸리기 때문에 실질적으로 의미 있는 만큼의 수를 내다 볼 수 없기 때문에, Alpha-Beta Pruning이라는 최적화 method를 적용시켰다.
* Alpha-Beta Pruning에서 alpha 값은 나에게 가장 유리한 상태의 점수이고, beta 값은 상대방에게 가장 유리한 점수이다. 즉, alpha 값이 클수록, beta 값이 작을 수록 좋다.
* 이때, alpha 컷과 beta 컷을 진행하여 minimax를 최적화 할 수 있다. alpha 컷은 상대 턴의 점수과 alpha 값을 비교하여 alpha 값이 더 클 경우 그 다음 나의 턴에 해당하는 node를 잘라버리는 것이다. beta 컷은 나의 턴의 점수와 beta 값을 비교하여 나의 턴 점수가 더 크다면 그 아래 상대 턴의 node를 잘라버리는 것이다.
* 예시로 생각해보자. 현재까지 수를 분석했을 때 기대되는 나의 점수는 3점이었다고 해보자. 이때, 그 다음 나의 턴의 한 경우에서 2점이 발견되었다면, 그 뒤의 경우들은 분석하지 않아도 된다. 왜냐하면, 그 다음 나의 턴은 상대에 의해 결정되는데, 상대는 점수가 나의 최소가 되는 수를 선택하기 때문에, 상대의 수에서 기대되는 나의 점수는 2 이하 일 수 밖에 없다. 즉, 이 가지에서는 현재 기대되는 나의 점수보다 더 낮은 점수를 얻을 수 밖에 없기 때문에 더 탐색하는 의미가 없는 것이다. 이와 같은 상황을 alpha 컷이라고 한다.
* beta 컷 또한 위와 같이 생각해보면 잘라야 할 이유를 알 수 있다.
* 이러한 minimax 알고리즘은 트리로 구현하는 것이 일반적이나, 구조체를 쓸 수 없기 때문에 트리를 구현하는데 어려움이 있기 때문에 재귀 함수를 이용하여 구현할 것이다.
* 우선 함수가 호출되었을 때, depth가 0 이하라면 prevScore을 그대로 반환한다. 아니라면, 우선 나의 턴 일때는 -1000을, 아닐 때는 1000을 optimalChildValue로 잡는다.
* 이후, 유효한 line들 안에서 아래 과정을 반복한다.
* 우선, board를 simulboard로 복사하고, line에 플레이어의 숫자를 마킹한다. 그리고 그 때의 노드 점수를 calcScore를 통해 얻는다.
* 이후, 더이상 simulboard에 유효한 선이 없다면, childList에 현재 노드 점수와 좌표를 push 한다. 아니라면, alpha\_beta\_minimax를 depth-1으로 호출하여 childValue 값을 받는다.
* childValue 값을 childList에 좌표 값과 함께 push 한다.
* 알파 컷/ 베타 컷을 진행한다.
* 모든 line에 대해 반복하였다면, childList에서 가장 높은 노드 점수 값을 찾아서 해당 좌표를 turn 함수의 좌표에 기입하고, 그 점수를 반환한다.

**4. 프로그램 실행 방법 및 예제**

**a) 로그인 및 컴파일, 실행 확인**

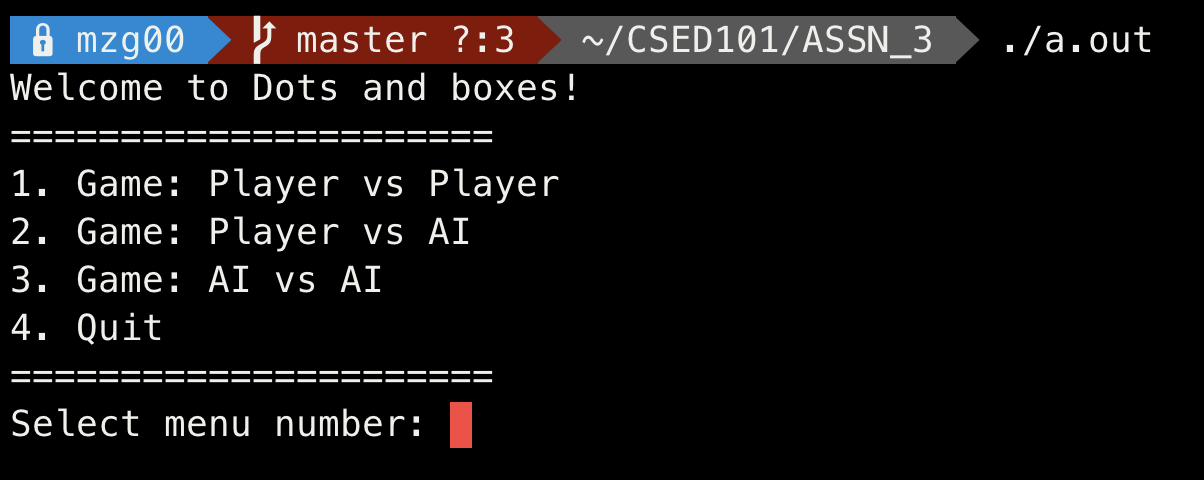


ssh 명령어를 이용하여 리눅스 계정에 로그인한다.

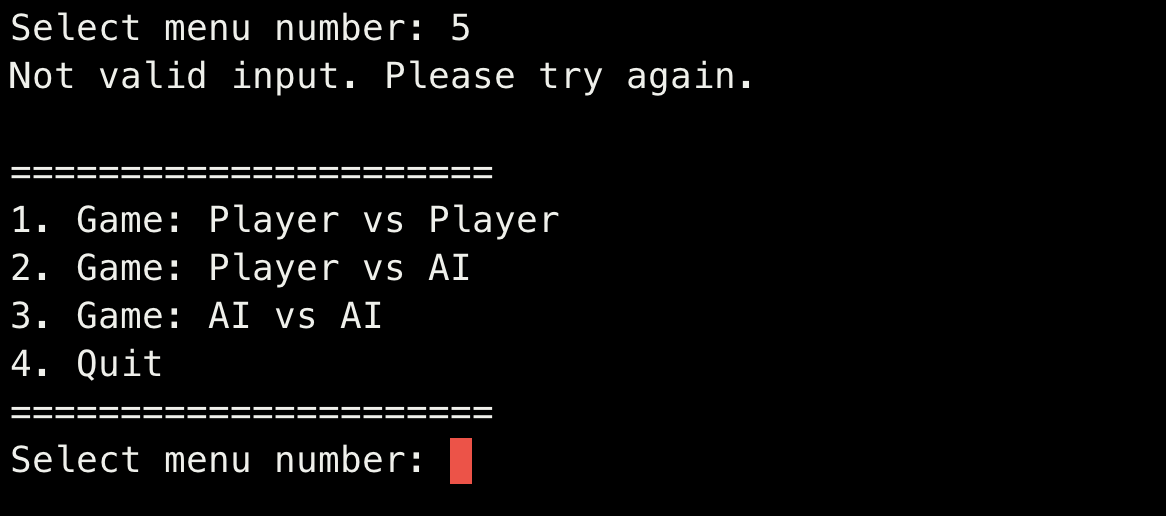


디렉토리로 이동하여 assn3.c을 컴파일 한다.

**b) 메뉴 화면**

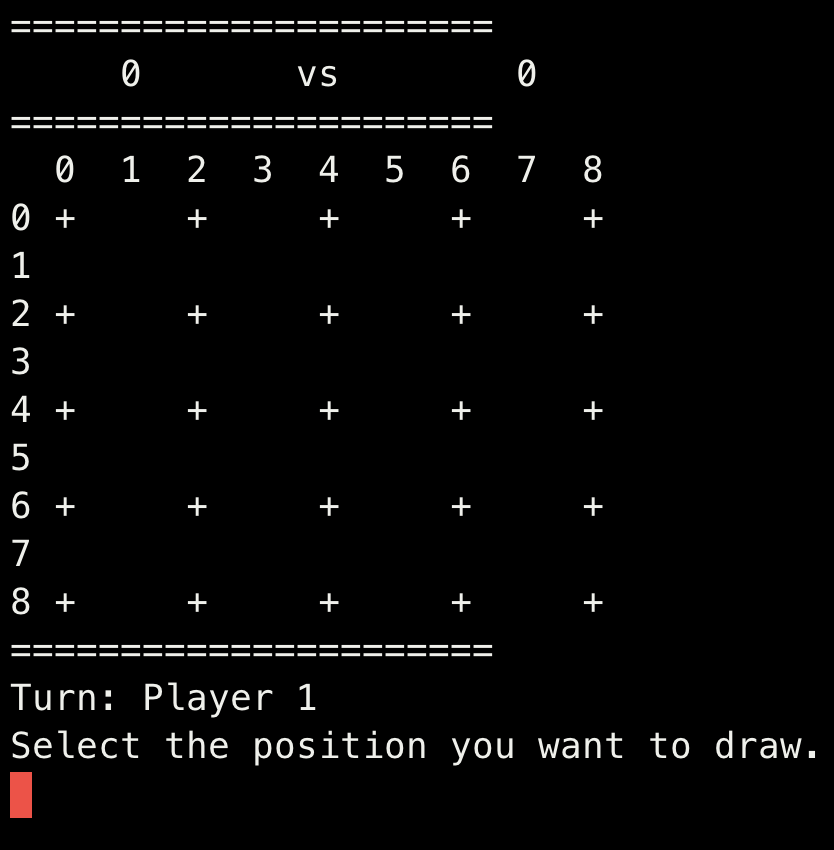
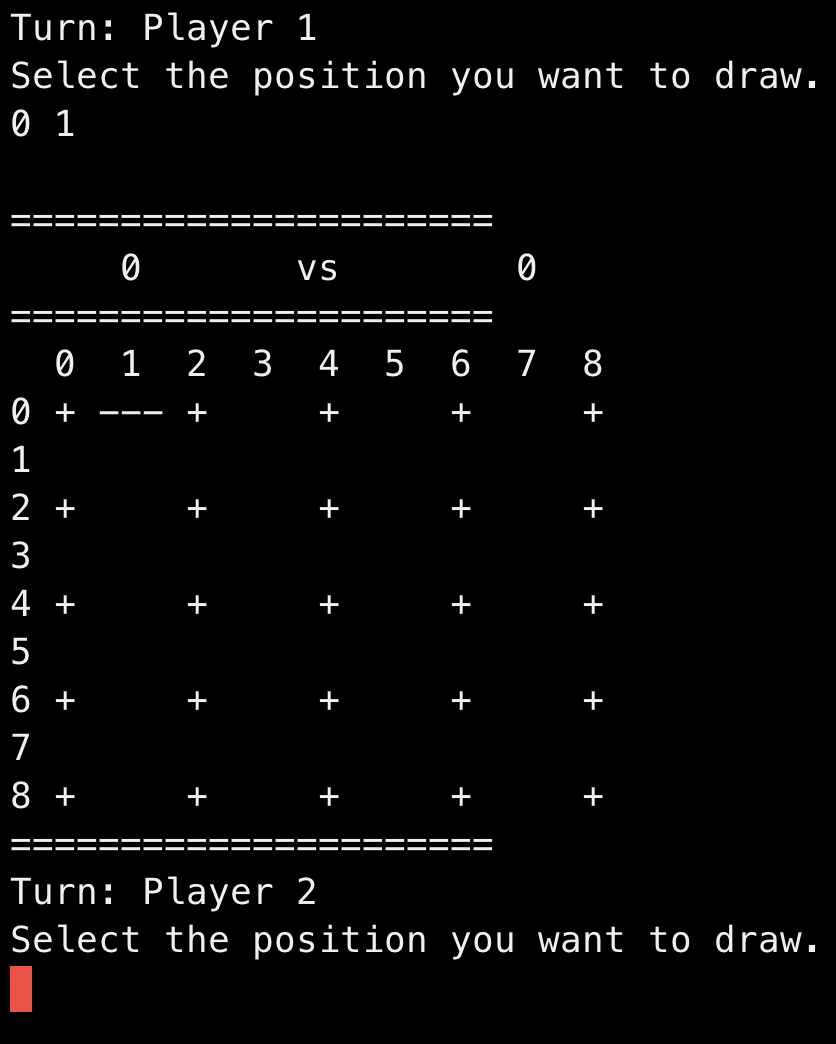
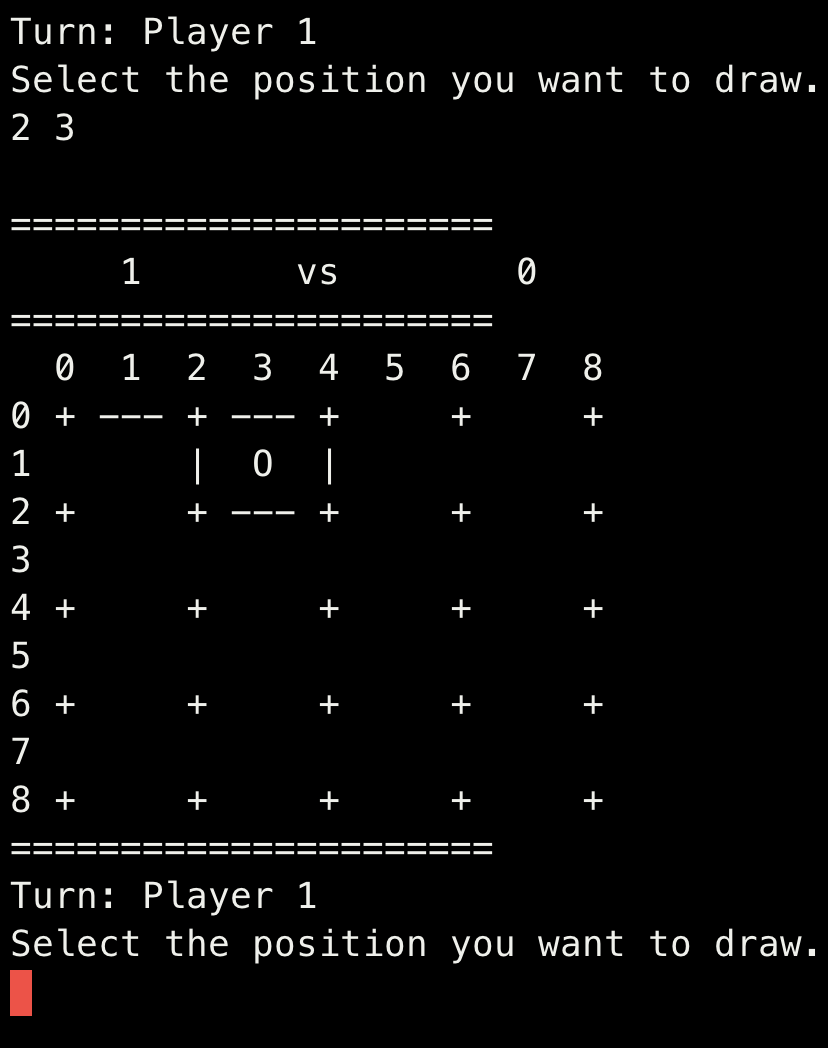


처음 프로그램을 실행하면 메뉴 화면이 출력되고, 입력을 대기한다.

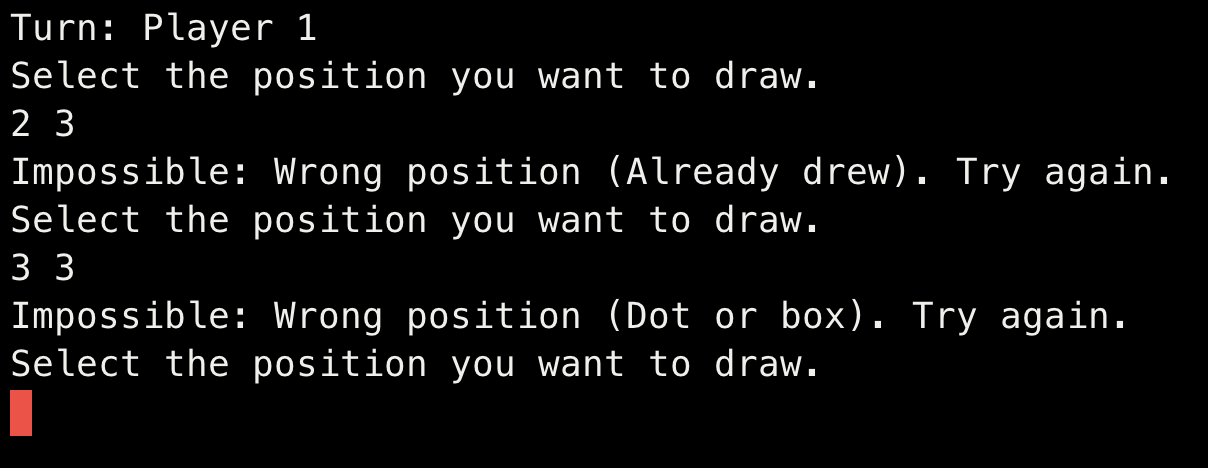


1~4 이외의 정수를 입력하면 올바른 입력이 아니라는 메시지와 함께 다시 메뉴를 출력하고 입력을 대기하는 모습을 볼 수 있다.

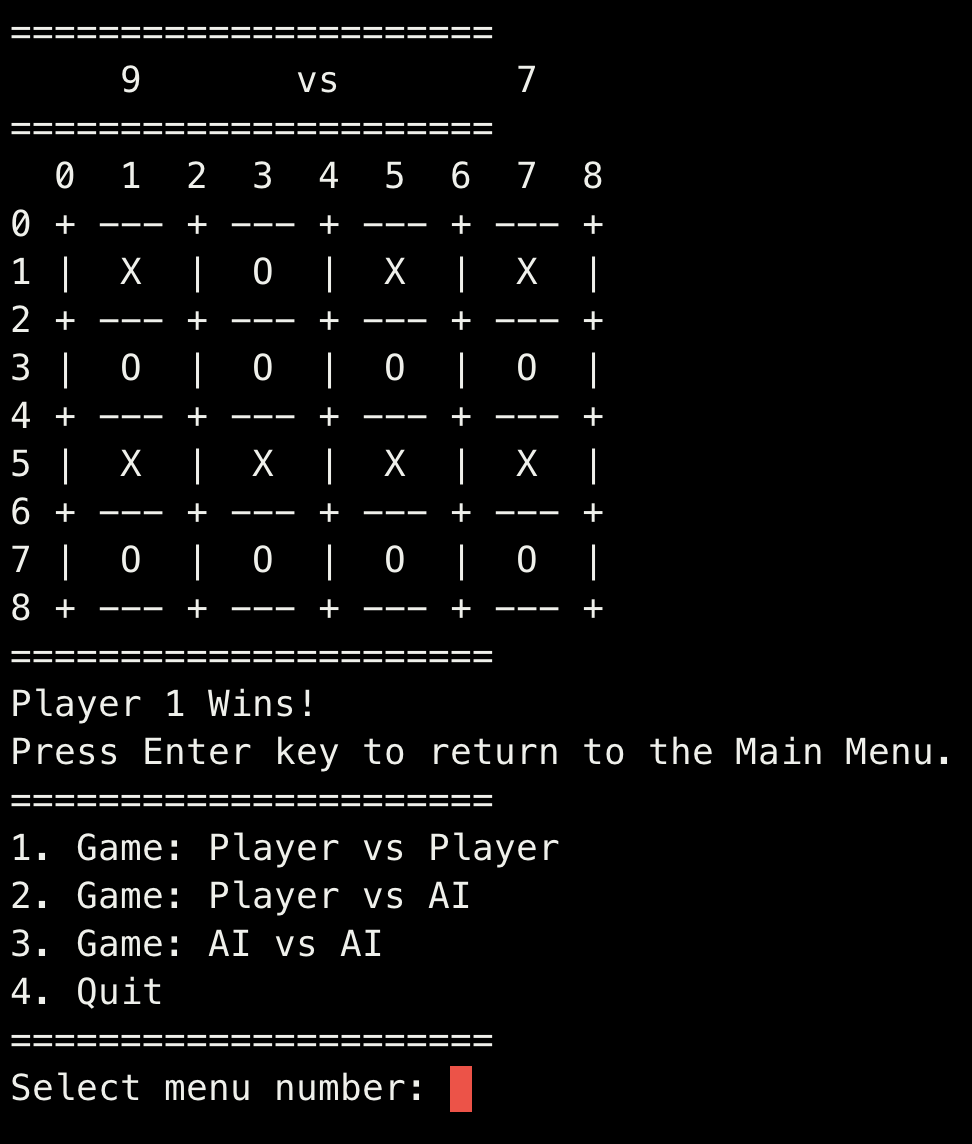
**C) PLAYER vs PLAYER**

1을 입력하면 플레이어 2명을 위한 게임 화면이 출력되며, 주어진 문제 상황대로 잘 출력하는 모습을 볼 수 있다.

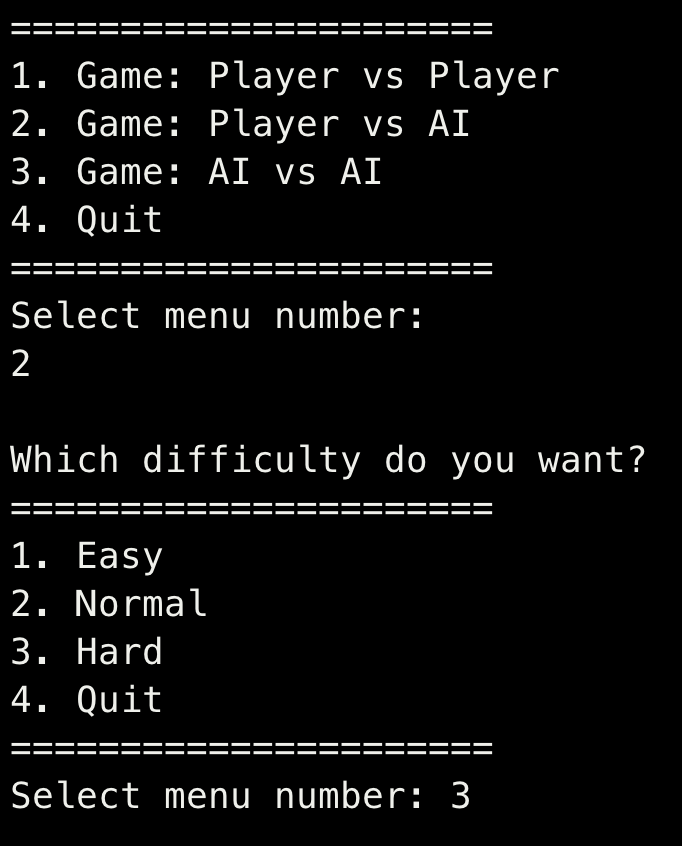
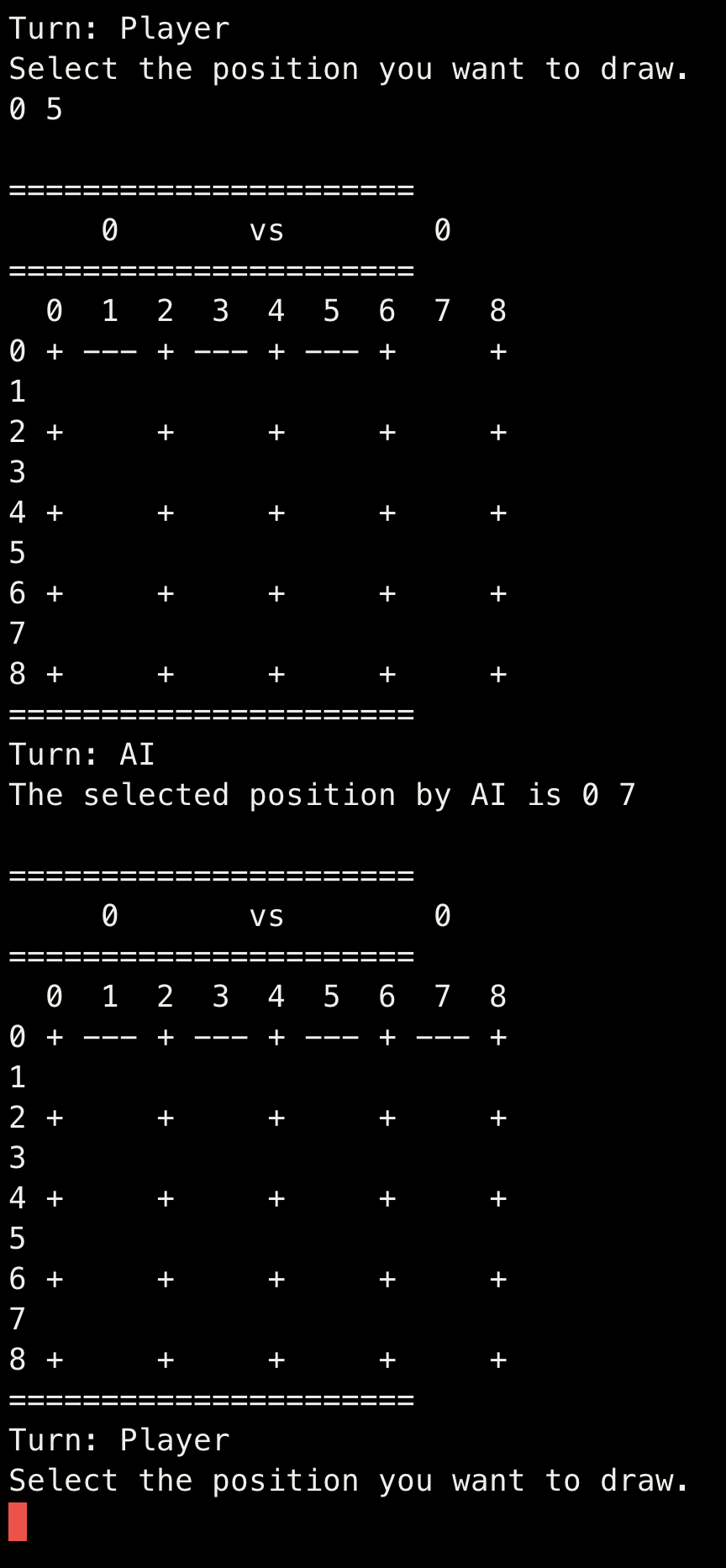


또한, 예외 처리도 상황에 맞게 처리되는 모습을 볼 수 있다.



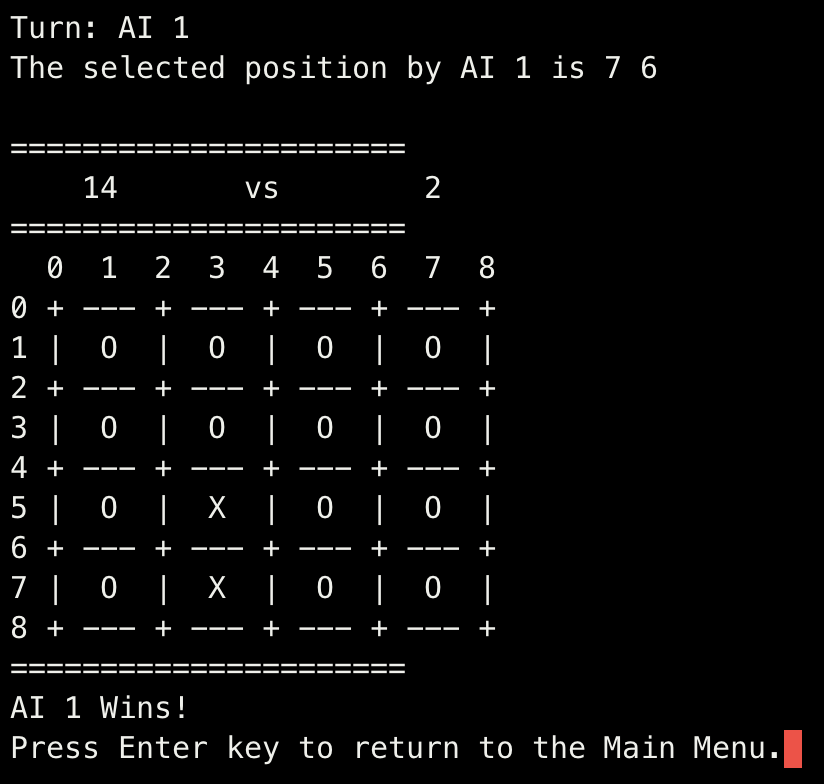
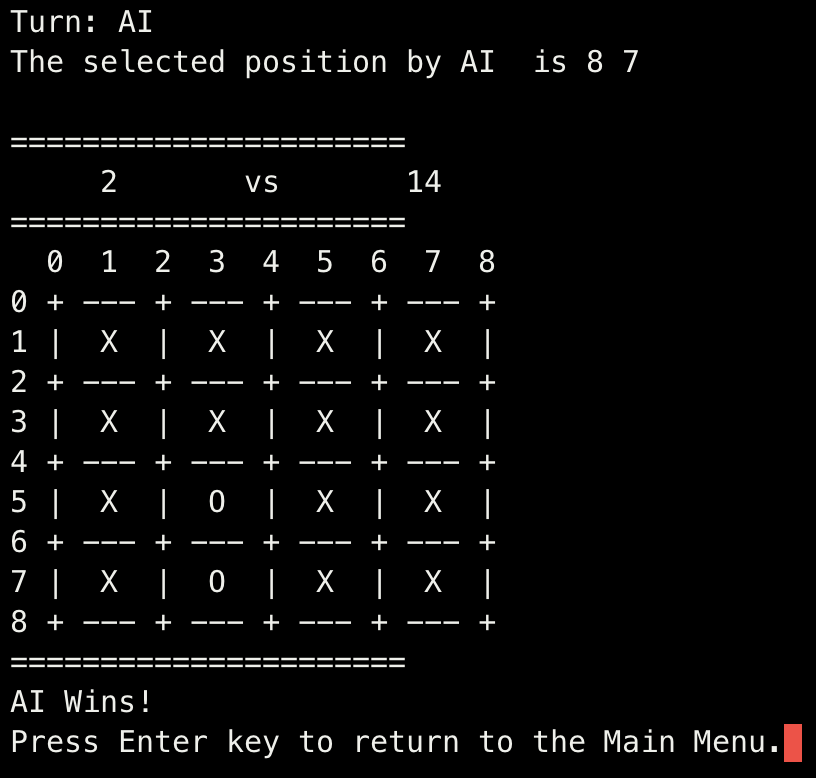
게임이 종료되면 승자를 출력하며, 엔터 키를 누르면 메뉴를 출력한다.

**D) PLAYER vs AI**

** **

문제에서 주어진 상황대로 잘 출력되는 모습을 볼 수 있다.

**E) AI vs AI**

문제에서 주어진 상황대로 잘 출력되는 모습을 볼 수 있다.

**5. 토론**

* 주어진 문제에서 Hard AI는 Alpha-beta pruning을 적용한 minimax 알고리즘으로 작동된다. 하지만, 이러한 간단한 게임에 비교적 많은 연산이 요구되는 이 알고리즘을 적용한 것은 오버스펙으로 느껴질 수 있다. 그러나 if-else가 많이 가미되는 수학적인 모델보다, 이러한 알고리즘의 구현이 훨씬 많은 수를 커버할 수 있기에 탁월한 선택이었다고 생각한다.

**6. 결론**

* 본 과제를 통해 2차원 배열을 어떻게 활용할 수 있는지 알 수 있었고, 특히 매개변수로 사용할 때의 주의점을 알 수 있었다. 또한, Hard AI를 구현하면서 재귀함수의 쓰임새에 대해 한번 더 알 수 있었다.

**7. 개선방향**

* 현재 작성된 Hard AI는 정적으로 메모리를 할당하는데, 꽤 많은 depth에 대한 탐색을 진행하면 메모리를 많이 차지할 위험성이 있고, 현재도 상당히 불안정하여 아주 가끔씩 Segmentation fault가 발생한다. 이러한 점을 개선하기 위해 dynamic allocating을 적용시킬 필요가 있다.