Coursework Report

Stephan Aldhous
40288816@napier.ac.uk
Edinburgh Napier University - Module Title (SET09117)

Abstract

This report will detail the inner workings of a game of English checkers (English Draughts). I will cover the structure of the game within the program including: the transition between turns and the stages that make them, the movement of game pieces and how it limits the options a player can make during a turn.

Keywords – Data Structures, Draughts, English, Checkers

1 Introduction

This project is to create a game of English Draughts. A game in which each player has twelve pieces, all placed diagonally from one another on an eight by eight checkered grid board. Each piece can only move diagonally. There are two type of pieces, one referred to as a Man and the other referred to as a King. Both pieces move slightly differently, the Man can only move diagonally forward relative to its starting side and the king can move in all four diagonal directions. There are no flying kings in English Draughts: [1]. If a piece has taken another piece it must continue taking pieces where possible until it is not possible to take any or it lands in the kings lane (kings row) and becomes a king. All pieces must take where possible.

The game operates using a state machine. State machines operate using a single current state which can change to a number of other states and will function differently depending on the current state. The only state that is apart from the game-play is the MainMenu state.

A single player can also play the game if they choose to play against an artificial intelligence. It is very rudimentary as it simple selects a random available piece and then picks a random available move for that piece.

2 Design

This section will discuss the architecture of the program and how it functions.

2.1 Program States

States within the program all inherit from a state called AppState. Appstate contains a confirmation method to ask if the user is sure of their answer. It also contains two other methods, one to update the state (UpdateState) and another to display a defined mass of text (ConstantDisplay), both of which can be overridden. In each state, the update method utilizes a switch statement to determine what to do given the input. This makes taking actions simple as the switch statement will jump straight to the default case if the input has not been defined in the statement. Each default scenario results in the method calling itself, resulting in restarting the state. Some cases also use the confirmation to ask the user if they are sure of the selection.

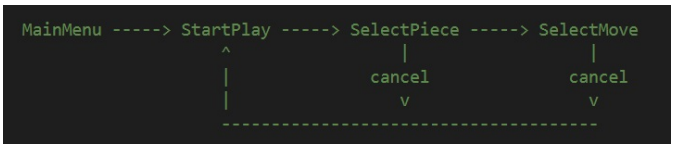

Figure 1: **State system** - A diagram of how the states work

The game-play states are StartPlay, SelectPiece and SelectMove. Within StartPlay the player determines if they want to play the turn, exit, undo the previous two turns or redo the next two turns. A player canceling their turn will be brought back to StartPlay. Choosing to play will take them into the SelectPiece state, in which the player is presented with a list of available pieces by board coordinates. The player then must enter a stated available piece and press enter. The player is then taken to the SelectMove state, in which the moves available is filtered to their piece selection. Each move is drawn on the board with a given number, which the player must enter in order to choose a particular move. The game manager then executes the move and moves onto the next turn (StartPlay) in which the next player must decide if they want to play. In the case of an artificial intelligence, the move will be taken the moment after the player has made their move.

2.2 Game Board

The game board itself is made up of a 2D array of BoardCells, each containing an array of string values representing the cell's content, a Unit occupant (can be null), coordinates of the cell, flag signifying if the cell is a King Row.

2.3 Determining Piece Selection

If a player has chosen to play their move then they are moved to the SelectPiece state. This state requests that the game manager works out the units the player has

available to move. This also involves the manager getting the moves for each of these available pieces as well, which is filtered down once a piece is selected.

In English Draughts, if an attack (jump) can be taken it must be taken, even if it results in the player being at a disadvantage. The game manager searches each of the player's pieces on the board checking for attacks first. This means that later if no attacks were discovered then the manager needs to check for normal moves, however if attacks were found then the check for normal moves does not need to take place. If a possible attack or normal move is found then the manager adds the board cell in question to a list of cells available for selection.

2.4 Selecting Moves

Once a piece is selected, the game manager then throws away the redundant selections and their respective moves so that only the selected piece's moves are left. This is done by checking if the starting cell of each move matches the selected cell. The SelectMove state will display these moves on the board for the player to quickly see and choose from. All the player must do is enter the move index and press enter to select the move. If the input is valid then the move selected will be executed, be it a normal move or attack move. The program state will then be set to StartPlay for the next turn.

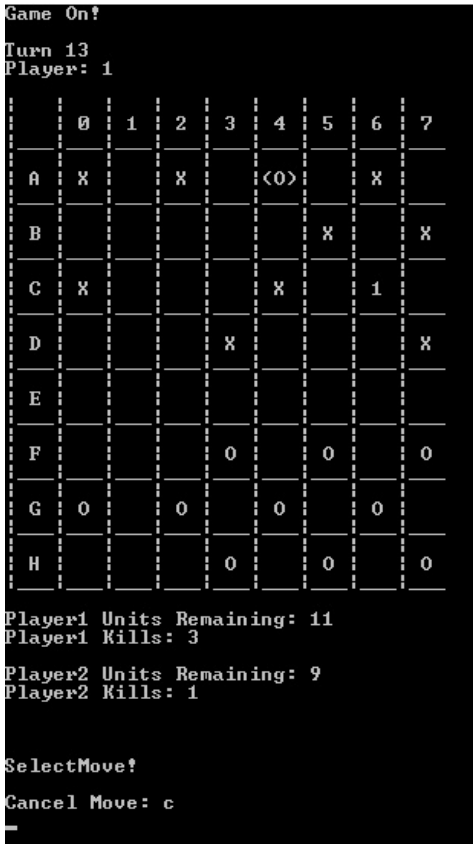

Figure 2: **Selecting a turn** - A screen shot of an attack option, the attack must be executed.

2.5 Finding Attacks

The game manager must have a way to find attack moves and normal moves. How does it find them? An attack move is to jump over an enemy piece and land in the cell beyond. This means that not only does the enemy piece need to be near the attacker, but also that the cell beyond must be on the board and not occupied. It must also be within the forward movement direction if the attacker is a Man piece. This shall be discussed in terms as if the attacker is a Man piece as a King does not need to care about the direction.

For a Man, there is only two move directions, forward-left and forward-right. The game manager will check both sides for attacks. The manager will check if the cell in the forward-right (or left on the other side) is on the board. If so then it will check if it is occupied and the occupant is an enemy. It then marks the cell as the defending cell. It then checks if the cell beyond is on the board and if so, if the cell is empty, then adds registers the cell as the finishing cell. If all of the above is satisfied then A possible attack will be registered within a list of possible attack moves. The same process goes for the forward-left direction and all four diagonals for a King.

2.6 Finding Normal Moves

Provided the game manager didn't find any attack moves then it will search for regular moves. The process is similar to attack checking. The difference is that if the cell to the forward-right for example is on the board and is empty, then it will register that cell as the ending cell and register the regular move in a list of possible regular moves. Again the same goes for the King in all four directions.

2.7 Multiple Attacks

It is possible that after an attack move is made, more may be possible. If this is the case, they must also be taken. The same process above is repeated for find new attacks if an attack was made, if none are found then the turn ends.

2.8 A Man becomes a King

It must also be made possible according to the rules for a Man to become a King. This occurs when a Man lands in what is known as a kings row, which is the furthest row on either side of the board. Once in this row the Man is then replaced with a King. This is done by checking if the landing cell, whether the move be and attack or regular, is in a kings row or lane. If this is so it also checks if the unit is a Man. If so, a method called ConvertToKing is called that removes the man from the player army and replaces it with a king unit. A king is signified by the brackets on either side of the piece symbol (Figure 2). A turn ends upon conversion even if an attack is possible.

2.9 Move Execution

Once a move is selected it must be executed. The execution is slightly different for each move. For an attack, the attacker occupies the finish cell as the start cell becomes empty and the defender cell's occupant is removed from it's player's army. The attacking player's kill count is also incremented. For a regular move the piece simply occupies the finish cell and the move is over.

2.10 Artificial Intelligence

A user can opt to play against an artificial intelligence. The intelligence however, is very simplistic. it find possible pieces and picks a random piece. It then filters the available moves and picks a random move. If it attacks it searches for other possible attack moves. If none are found it finishes the turn. This can make both easy and difficult to beat as it makes bad moves but can also catch the player off guard making a random move ruining a player's plan.

2.11 Game History

This is the only area where a simple list would make it harder to implement, instead the data structure used was stacks. There is a Future stack and a Past stack. When undo is called that Past is popped and pushed the Future stack until the current turn has decremented by two, and redo does the opposite. The stacks are composed of objects called HistoryItems, each item contains data for both players being: a list of army unit locations, a list of army unit types, and a kill count. They also contain the current turn value. Each turn that is taken adds an item to the history and also erases future moves as they are no longer accessible.

3 Enhancements

Given more time, Flying Kings could be implemented. A Flying King is a King that can move any distance in a given direction, allow the King to attack pieces at any distance as long as they are within a diagonal line. This also means that multiple attacks can be made easily since the King could jump long distances in successive attacks making a Flying King an adversary not to be taken lightly. The disadvantage of implementing a Flying King is that it possibly makes the King piece perhaps to powerful, as the first player to get one can easily sweep most of the opponent's pieces in a single turn.

Another thing that could be implemented given more time is a more calculating artificial intelligence. The current artificial intelligence simply moves at random and can be easy to beat, however this randomness can catch a player off guard quite easily. The reason this would take substantially more time is that the way the system is setup it isn't possible to check moves ahead as the game only checks for more moves once an attack is made.

4 Critical Evaluation

Using a state machine was a good choice as it made turn playing simply to implement as it breaks a turn up into easy stages. It also saves on embedding many checks during a turn into man y switch or if statements allowing each stage to be uncoupled from lots of other pieces of logic. It also allowed for the state display to be determined nicely for each state.

The way multiple attacks were setup, were done so poorly. In Draughts pieces are not taken of the board until the turn is finished. So executing moves as soon as they are selected, although does the task, does the task wrong. It also makes predictable paths much more difficult to implement meaning the artificial intelligence was lacking. This system also makes Flying Kings virtually impossible, since after the King jumps a piece, checks must be made for other attacks far beyond the taken piece, so the King can be placed correctly.

5 Personal Evaluation

During this project I have learned that successive searching is much harder than I initially thought. It is similar to a binary tree, however not every option has more than one branch. Unfortunately time constraints made this difficult to implement so a lesser approach was used in its place.

Another challenge was deciding what pieces could be used. This was overcome by have three lists: Attacks, Moves and CellsAble. If Attacks is empty the manager than looks for regular moves to add to Moves. If a move of any kind was found then the starting cell (the cell the piece occupies) then that cell is added to CellsAble provided CellsAble does not contain the cell in question. CellsAble is the relayed to the player during their turn. Once a cell is selected, all moves are filtered until only the moves with the cell as their starting cell remained.

References

[1] "English draughts rules." https://en.wikipedia.org/wiki/Draughts. Accessed: 2017-11-09.