FlicFlac AI Implementation

Key: Pseudo Code
Structures/Code to be found in flicflac/shared/src … unless client specified

01: Initial Role

On start up the AI opponent activates when the opponent's name is "***". This forces the bootstrap role of the AI to be the "Initiator", according to the pseudo code below: (Your name is always alpha-numeric.)

```
if NameOfAI < YourName then
  AI = Initiator // this player entity attempts to make the connection in peerJS
else
  AI = Responder // this player entity listens for a connection on peerJS
end if
```

02: Initial Parameters

The parameters for the AI entity (check PlayerParams.PlayerParams), should be taken from your parameters, except for "Turn Time" and "Captors Time", which the AI can ignore.

An initial FlicFlacGameModel can be created using ...

```
val model = new FlicFlacGameModel()
```

… then populate various fields according to PlayerParams.

03: Maintaining "Player Names" and "Piece Type" during communications

Any decoded inbound peerJS message will be an encoded version of the remote player's FlicFlacGameModel and some fields must be adjusted before any processing can occur.
Use FlicFlacGameModel.ConvertRxGameModel() to achieve this.

04: Initial Bootstrap State Machines

The first AI goal is to complete the bootstrap process. The "Initiator" state machine and "Responder" state machine are implemented in client/src/SceneParams.scala. The AI needs to mimic the initiator state machine only, in order to transition through the boot strap game states (FlicFlacGameModel START_CON1 … START_CON8) to arrive at game state CYLINDER_TURN. These state machines are tied to Indigo and PeerJS so inclusion as a shared resource was prohibited. The AI maybe playing CYLINDERS or BLOCKS, this is randomly selected and not tied to the initial bootstrap role of "initiator".

05: Establishing the Hex Board for the AI

Use the following code and the board size values in HexBoard.HexBoard to establish the hexagonal board. This will give the correct size, coordinate system, coloured hexagons and extra home hexagons.

```
val hexBoard = new HexBoard()
hexBoard.forge(n) // where n is the size of the board
```

Please refer to the four accompanying PNG files that show each coordinate system in further detail.

06: Summoning the Pieces for the AI

At the start of the game the piece types are randomly selected. If the AI is CYLINDERS, then the AI has the first turn. Use the following code to establish the 12 playing pieces (in the model) at their starting positions. The functions to manipulate these pieces are supplied in the shared classes Piece & Pieces

```
val startingPieces = model.pieces.summonPieces(hexBoard)
val newModel = model.copy(pieces = startingPieces)
```

07: The AI Turn

At the start of an AI turn, process the incoming game model by calling ...

FlicFlacGameModel.ConvertRxGameModel()

Then consider the strategic value of the various combinations of moves for the 6 AI pieces. These moves can be accumulated by selecting each piece in turn …

- Piece.setSelected(true)

… then acquiring the valid set of moves for this piece …

- Spots.calculatePossibleMoves()

… then deselecting that piece

- Piece.setSelected(false)

The maximum number of possible move positions for one piece is 25. (This occurs when the piece is on a central black hexagon, unhindered by any other piece.) Potentially the maximum simultaneous combination of moves to consider might be 25 to power 6, However, the possible movements are restricted by the edge of the board, the other AI pieces and opponent's pieces. So the final total is much less. Remember to flip a piece if moving to an adjacent black hexagon. Also, additional logic is required to ensure the AI does not place a piece on top of another of it's own pieces that already has a new position.

The strategic value, (this might be the number of captures achieved in this turn), of any one combination of 6 simultaneous moves could be determined by calling …

- Melee.combat()

… then …

- Melee.detectCaptors()

Captures achieved is only one possible way for the AI to evaluate each possible move combination, other possibilities may be better.

When the AI has determined the optimum movements for it's 6 pieces, it should implement the following pseudo code to end the turn:

```
newScore = pieces.extraTurnScoring()
captors = Melee.detectCaptors()
if (captors.isEmpty) then
 pieces.newTurn()
else
  newPieces = Melee.rewardCaptors()
```

If there were captures, then the initiative is still with the AI.

Finally, the AI could try to iterate more than one move ahead to find the optimum move combination for this turn. Under these circumstances, the AI would need to consider all possible move combinations for the next turn of the opponent etc.

oooOOOooo