**The Wooden Man Presentation**

[Title slide]

Ladies, Gentlemen, Robots, and non-binary organisms, may I present The Wooden Man.

[Picture of robot – screen showing a “smiley” face?]

Not as heavily armoured, or as powerful, as the Iron Man, but, being made of recycled kitchen worktop, and with a Raspberry Pi instead of a nuclear reactor at his heart, he is considerably more environmentally friendly and likeable. Having said that, his mission is to beat you into submission at a range of board games, and the sharp corners aren’t that cuddly.

Here you can see that Woodie has a webcam, allowing him to see, a screen through which he can communicate, and a robot arm to allow him to interact with the world. Here on his chest is hidden the secret of his powers: a Raspberry Pi.

[List of objectives/challenges]

This project grew out of a challenge set by the West Yorkshire police to the five universities in the region. This boiled down to “we’ll give you a Raspberry Pi: do something interesting with it”. The Pi is a single-board computer whose chief merits are that it is small, cheap, and well equipped for controlling external hardware, so we wanted to do something consistent with those strengths.

The aim of the project was to create a robot which could actually play board games against a human opponent: detecting the actions of that opponent, and reacting with its own moves in real life. We chose games with similar playing mechanisms, such that robot control mechanisms would be largely shared, but which posed substantially different AI challenges. Tic-tac-toe is a very simple game, which can be solved completely in real time, even by a modest computer like the Pi. Nine Mens Morris is almost an extension of tic-tac-toe, but while it is technically solved (it should be a draw), it is too complex to solve in real time, so a more sophisticated, look-ahead and analyse, approach is necessary. It also has the merit of being relatively little known by the general public, giving the robot a better chance of beating you! Draughts is of comparable complexity mathematically, but has very different gameplay, and is better known.

We chose this project because it would bring together a range of technical challenges:

1. Woodie needs a vision system able to identify which game is being played, identify counters as they are placed, and convert the images into a board position. The images also need to be remapped to allow calculation of the position of objects in 3D space.
2. We wanted Woodie to be able to interact directly – in other words, to be able to play the games himself. Thus he needed a robot effector able to manipulate pieces and place them at specified points in space. Preferably the RIGHT points...
3. AI players would be needed to evaluate game positions, formulate possible moves, and ensure valid gameplay.
4. A core “operating system” would need to coordinate the different units: for example, getting a board position from the vision system, check its validity, and pass it to the AI. Once the AI returned the desired move, it would have to calculate what that required in terms of movements of the robot arm, and pass them to the motor controller.

[Vision System]

In the spirit of the Pi, we wanted to keep the hardware fairly cheap and simple, so we opted for a single webcam for the vision system. Since we wanted the robot to “meet you across the board”, we didn’t want to build a superstructure allowing a vertical view of the board, so we have mounted the camera on top of the monitor, giving it an oblique view of the playing surface. We need to be able to map this view into three dimensional space, to allow the robot to interact with it. Some parts of this problem we have solved mechanically, for example by having a horizontal board at the same level as the base of the robot, and some we have solved in software, using openCV for image transformation and detection of lines and circles. As you can see in this right hand image the computer’s view of the board looks like it was taken from overhead: it is also highlighting the circular counters it can see. In the screengrab on the left, we can see that it found 4 line intersects, allowing it to deduce that this is the tictactoe board. We can see that initially it could see the human’s opening move, and then later it recognised where it had itself played. This information is then passed to the main control program.

[Robot arm]

Initially we looked at using a commercial robot arm, but affordable ones were too small, and anyway it seemed more fun to build one. Our early research suggested it should be possible to drive a Lego Mindstorms system from the Pi, and the advantage of this is that the motors can both drive the movements, and act as sensors of their position. A big enough Lego arm proved difficult to build, because it flexed too much, so a wooden one was constructed. The base is a heavy turntable incorporating a ball race made of marbles, which can be turned by a motor.

Rubber bands try to pull the arm into a vertical position, and the two motors mounted on the base pull against them to flex the arm. This means that a single motor for each joint can both flex and extend the arm remotely.

The two motor-controlled hinges in the arm allow us to control the reach and height of the end of the arm. Effectively the turntable angle and the reach of the arm form polar coordinates over the playing area. To manipulate the counters, an electromagnet was mounted so that it would always hang vertically: this could be switched on and off using a fourth Lego motor.

[AI Approaches]

The first thing you need to know in order to play a game is the rules. We can encapsulate these in a routine that determines whether a potential move is valid. For example, in draughts, you can simply examine whether the squares diagonally ahead of pieces are free, or, if it is occupied with an enemy piece, whether the space beyond is free, in which case we need to note that a jump is possible, and no non-jumping moves are allowed. For kings, backward moves also need checking. We can then use this routine to try out all possible moves. But how do we choose between them?

All games were approached by using a minimax algorithm to decide what the robot should play. In this approach we look ahead a few moves, trying all legal combinations, and evaluate the resulting positions, bearing in mind that both players will be trying to maximise the score for themselves, which minimises it for the opponent. For tictactoe the game is simple enough that the analysis can always reach the end of the game, so the robot should never lose.

For more complex games it was necessary to develop an evaluation algorithm that allowed the program to assess how strong its position was. This was done by some initial “common sense guesswork” of aspects of the game position and their importance, and refined by allowing algorithms to compete against each other. For example, in drafts the number of pieces, and especially the number of kings, is important - but a piece with a free route to becoming a king is more valuable than one about to be taken, for example.

Once you decide what is important, we can optimise the weightings using various approaches, including automated, self-learning methods such as getting the computer to play various algorithms off against each other, gradually varying the parameters. This turned out to be a complex and interesting part of the process, which we can discuss later, if people want to.

Much of the process is the same whatever game you are playing, so we should be able to share the code for this. Much of the core logic was implemented as an abstract class: the robot can use basically the same core code to play any zero-sum game, as long as you supply it with specific algorithms to test for a valid move, and to evaluate a position, for each game. This is quite efficient, and also makes it relatively easy to extend the range of tasks the robot could be used for.

[Holding It Together – system diagram?]

So we come to the “main” program: the bit that ties it all together. Like most managers, it spends most of its time sitting in its office playing executive games. When the vision system calls and says “I’ve got a tictactoe board position for you”, it takes it, and sends it to the AI system, with a request to work out the response. What it gets back is “get the counter at position A and move it to position B”. It then calls a conversion system, which returns a message saying “Position A corresponds to this set of parameters for the robot arm, and Position B corresponds to that one”. Finally it creates a set of instructions for the robot arm (these commands are slow to transmit to the EV3 controller, so we send a whole bunch at once). First the arm needs to move to where the counter currently is, then it needs to pick it up, then it moves to where it is going, drops it, and finally moves back to its resting position. Then the control system can go back to Office Golf until the vision system chips in again.

[Future Perspectives]

Where else could we take the project?

We could expand the range of games Woodie can play: as I mentioned earlier, this is a relatively simple process, because of the flexible way his core systems are encoded.

One game we wanted to play was connect-four, but the robot arm in its present configuration would be unable to handle this. It would be fun to build a better arm, with more degrees of freedom in its movement. This would mean moving away from the current system using Lego motors, and probably incorporating small motors in the arm itself. A gripper, rather than just an electromagnet, would be a more flexible and powerful way of manipulating the world.

Similarly, it would be nice to take the motor and sensor control “in house”, as it were, rather than delegating them to the Lego EV3. One of the strengths of the Pi is the presence of lots of input/output pins that could be used for these.

It would be possible to add a second camera, to provide stereo vision. This would be significantly harder to interpret, but would give Woodie true depth perception.

[Discussion Points]

That concludes this presentation, but we are happy to answer any questions. A few things you might like to discuss in more detail are:

Communication between the Pi and other devices: the webcam, and the Lego EV3 controlling the robot

The vision system and how it works

Training the AI

What would need adding to take control of the robot in-house, on the Pi, and/or to enhance the function of the arm

Interesting points from training

Having AI look only for the BEST move results in totally predictable play – which is easy to beat. It seems much better to allow it to accept one of the EQUAL best. Also helps it not get stuck in a see-saw of moves.

I wonder if I tend to select for weightings that lead to a lot of draws, to avoid losses? What do you really want?

I tried random weightings, which sample a large space quickly. I also tried an evolutionary approach, with either lots of parameters changing a little (too many changes means new version is nearly always less fit), or changing only one parameter (more sensible, but pretty slow). These routines ran for only one game, with “winner stays on” – but I’m concerned that weaker routines would still win sometimes, throwing away good parameter sets.

Moved on to try targeted evolution: I would choose parameter changes to try, and let it run for a number of games (up to 200 overnight!). Illustrates superiority more clearly. Allows you to work through: optimise phase 1 play, then phase 2, then phase 3.

Could go for a compromise: maybe automate playing 10 games for evolutionary routine.

Even lookahead of 2-3 (which runs pretty fast) seems to be a pretty good discriminant of the evaluation parameters, at least in broad terms. Finer tuning done with a lookahead of 3-4.

I’d like to think about ways to let it spend some extra time ONCE in a game, to try to break a deadlock with repeating moves, or in the endgame to find a finish.