Zoeken, Sturen en Bewegen 5082ZOSB6Y
Verslag Groepsopdracht
Implementatie van 3D Tic-Tac-Toe voor de UMI RTX robot arm

Janosch Haber 10400192
Michiel Boswijk 10332553
Joost Hoppenbrouwer 10334564

5 juli 2013

Samenvatting

In dit verslag wordt het implementatieproces van een 3D Tic-Tac-Toe variant beschreven. Deze variant bouwt op op het idee dat stenen in de twee bovenste lagen alleen kunnen worden geplaatst als hier onder ook al stenen liggen die deze posities ondersteunen. Hierdoor is het mogelijk deze implementatie door de UMI RTX robotarm te laten spelen. Welke redeneringen, ontwikkelingen en aanpassingen in dit proces werden gedaan zal hier worden toegelicht.

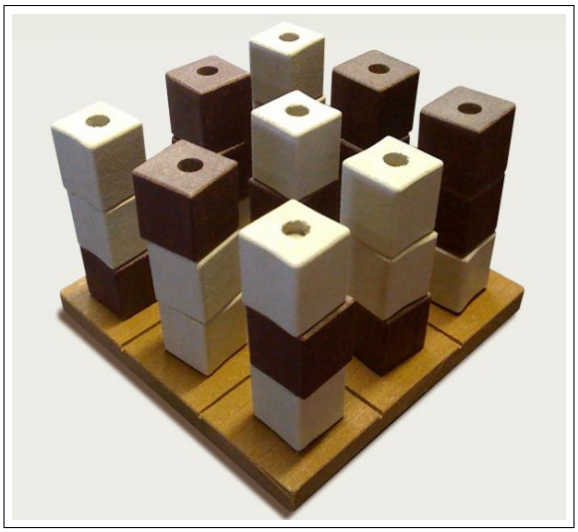
Inhoudsopgave

Inleiding

Dit verslag is geschreven voor het vak "Zoeken, Sturen en Bewegen", het laatste projectvak van het eerste jaar van de Bachelor Kunstmatige Intelligentie aan de Universiteit van Amsterdam. In de laatste week van dit vak werd groepen de ruimte geboden aan de slag te gaan met een zelf gekozen robotica project. Dit verslag is het resultaat hiervan.

Het Spel

In dit verslag wordt een variant van het bekende spel Tic Tac Toe (Boter, Kaas en Eieren in het Nederlands) gepresenteerd, dat gespeeld kan worden met de Umirtx robot arm. Om het wat uitdagender te maken is gekozen voor een drie dimensionale variant. Er bestaan al meerdere bekende 3D varianten van het spel die met een grafische interface gespeeld kunnen worden, maar de variant die hier wordt gepresenteerd is minder bekend. Het gaat om een versie van het spel waarbij er alleen toegang is tot een opvolgende laag als er onder die laag al en stuk is gespeeld. In het figuur hiernaast is te zien hoe deze versie eruit ziet (merk op dat de oorspronkelijke 'x' en 'o' zijn vervangen door blokken van twee verschillende kleuren). Er kan op drie manieren een punt verkregen worden: drie op een rij horizontaal, verticaal, en diagonaal. Elk van deze manieren kan in een drie dimensionale ruimte verkegen worden. Aangezien deze variant doorspeelt tot wanneer alle stukken zijn gebruikt kunnen er dus meerdere rijen van drie verkregen worden per speler. De speler die aan het eind de meeste sequenties van drie heeft weten te behalen wint.



Een voorbeeld voor de 3D Tic-Tac-Toe implementatie (2)

Het Onderzoek

Als onderzoeks vraag is gekozen: Hoe kan een slim 3D Tic Tac Toe spel geïmplementeerd worden die werkt op de Umirtx robot arm?

Het belangrijkste uit de onderzoeksvraag is dat de kunstmatig intelligente tegenstander slim genoeg is om het spannend te houden voor de speler, en dat deze software werkt op de simulator van de Umirtx en dus ook op de arm zelf. Om deze reden is de volgende hypothese gekozen:

Middels een puntensysteem kan een slimme AI geïmplementeerd worden en middels een Inverse Kinematica module kan dit programma draaien op de Umirtx robot arm.

Zoals de hypothese aanduidt is de voorspelling dat er met de beschreven methode een AI kan worden geïmplementeerd die slim genoeg is om een menselijke speler te kunnen verslaan. Ook is de voorspelling dat dit spel zal werken op de Umirtx robot arm. Dit laatste omdat de Inverse Kinematica module van de robot arm al in de week ervoor was geïmplementeerd. De module moet alleen aangepast worden om voor 3D Tic Tac Toe te werken.

Er is al veel software geschreven die het spelen van de originele versie van Tic Tac Toe tegen een slimme AI mogelijk maken. De AI wordt meestal simpel geïmplementeerd door middels één grote if else statement **alle** mogelijkheden langs te gaan (1). Dit wordt echter een onmogelijke taak wanneer er meerdere dimensies in het spel zijn. Er zijn ook versies te vinden van een drie dimensionaal Tic Tac Toe spel. Deze varianten stoppen meestal zodra een speler zijn eerste rij heeft verkregen, en lopen in meer dan 2000 regels code ook alle mogelijke opties af, of zijn goed beschermd en dus niet online te vinden, zoals bijvoorbeeld de MSN variant (3). Om deze reden is de keuze gemaakt om een eigen strategie te bedenken voor het schrijven van de AI. Dit zal later in het verslag worden beschreven.

De opzet van het onderzoek was als volgt:

1. Inlezen in bestaande strategieën
2. Verdelen onderdelen
3. Individueel werken aan onderdelen
4. Voortgang bespreken
5. Testfase & Debuggen

De laatste drie punten werden herhaald tot het doel bereikt was.

Relevantie

Dit onderzoek is maatschappelijk relevant omdat deze versie van het spel een meer uitdagende versie van het originele spel is. Mensen die het originele spel kennen waarderen het spel minder omdat er weinig diepgang in zit. In de huidige versie zit meer diepgang en zal dus een positieve variant zijn op het origineel en mensen stimuleren meer na te denken bij het spelen van een zet.

De wetenschappelijke relevantie ligt in de relatief makkelijke methode voor het implementeren van de AI voor een slimme tegenstander. Deze methode kan gebruikt worden bij spellen in de toekomst die een snel algoritme vereisen dat weinig rekenkracht kost.

Materiaal

Het materiaal dat was gebruikt waren Dell desktops met i3 processoren. Dit waren de computers beschikbaar in G0.10 van het Science Park Amsterdam. Het besturingssysteem op deze desktops was de Linuxdistributie CentOS. Voor het schrijven van de code werd de editor gedit gebruikt, en voor het runnen van deze code werd java jdk 1.6 gebruikt. Deze versie van java was reeds geinstalleerd op alle desktops.

Methode

Implementatie van de computertegenstander

In tegenstelling tot een eenvoudig twee-dimensionaal 3x3 Tic-Tac-Toe speelveld levert de geplande 3D implementatie en rij problemen voor het genereren van computerzetten:

1. Het doel van het spel is niet meer slechts het verkrijgen van één winnende rij en het voorkomen van een overwinning van de tegenstander door het blocken van zijn mogelijke rijen. Door het puntensysteem kan het soms meer opleveren om de tegenstander gecontroleerd enkele rijen te laten verkrijgen en daardoor zelf kansen op meerdere andere rijen te genereren. Er kan dus niet van een öptimale tegenstander"worden uitgegaan omdat sommige tactieken pas naar en groot aantal stappen zichtbaar worden en niet meer volgens actie-reactie kan worden geredeneerd.

2. Niet alle locaties op het veld zijn altijd bereikbaar. Door het feit dat de steentjes in de tweede en derde laag door steentjes in de lagen daaronder moeten worden ondersteund zijn per zet maximaal 9 van de 27 velden berijkbaar. Ook hiermee moet rekening worden gehouden bij het genereren van een strategie.

3. De verschillende mogelijke rijen kunnen praktisch niet door permutatie en aansluitend filteren worden gegenereerd. Hoewel er een slimme mogelijkheid bestaat om mogelijke rijen te bepalen (Neem de som van de coordinaten van de 3 velden van een eventuële rij en ga na of de modulo gelijk is aan 0) is het praktisch niet mogelijk om een lijst met mogelijke rijen aan te maken.

Uitgaande van deze lijst en de overweging dat het spelen tegen een perfecte computertegenstander (de beginnende speler heeft een winnende strategie) subjectief gezien niet leuk is, werd het besluit genomen de computerzetten niet via een zoekruimte maar door een regel gebaseerd opstel te laten bepalen. Eerst werd overwogen de gegeven PROLOG bestanden met een variatie van de AL-0 adviestaal voor het oplossen van een schaak endgame aan te passen voor het vinden van een 3D Tic-Tac-Toe strategie (deze bevat ook maximaal alleen 14 zetten). Omdat dit gedeelte echter geen recursie vereist en de communicatie tussen PROLOG en Java alleen via het aanmaken en uitlezen van textbestanden mogelijk is, werd uiteindelijk het besluit genomen om ook het regelsysteem in Java te implementeren en zo een volledig zelfstandig programmapakket te ontwikkelen.

Het idee van de MoveGenerator is het volgende:

1. In het hoofdgedeelte van het programma wordt een nieuw spel met computertegenstander aangemaakt en een speelboord gegenereerd

2. Als de computer aan de beurt is wordt het spelbord aan de MoveGenerator doorgegeven

3. De MoveGenerator genereert een lijst met alle mogelijke zetten (alle posities die door stenen in de lagen eronder ondersteund worden of zelf in de onderste laag liggen). Dit zijn er altijd maximaal 9

4. Voor iedere van deze 9 posities wordt bepaald in hoe veel en welke rijen deze positie ligt (minimaal 4, maximaal 14 in het midden)

5. Deze rijen worden afgegaan en de gevonden speelstenen opgeslagen. Hierdoor kunnen ze worden geëvalueerd op hun taktische waarde

6. Afhankelijk van de configuratie van de rij wordt nu een bepaalde score toegekend. Deze scores worden bij elkaar opgeteld voor de gegeven positie

7. Is de score voor de gegeven positie hoger dan de score voor een eerder geëvalueerde positie, dan wordt de momentele positie opgeslagen als favoriet

8. Zijn alle posities doorlopen, dan wordt de beste zet doorgegeven aan de Board class om de spelsituatie aan te passen.

9. Door het oproepen van de PP (PathPlanning) en de IK (InverseKinematics) classes wordt de gegenereerde move doorgegeven aan de UMI RTX robotarm om de coordinaten in een beweging om te zetten.

10. De MoveGenerator eindigt en het hoofdgedeelte van het programma gaat verder (toont de zet en vraagt voor een zet van de menselijke speler)

In iets meer detail werd de MoveGenerator als volgt geïmplementeerd:

De mogelijke posities worden bepaald door een lijst aan te maken van alle velden die niet al door een steen zijn bezet en gesteund worden door stenen in de lagen daaronder (of zelf in de onderste laag liggen). Deze lijst wordt doorgegeven aan de methode getMoveScore, dat deze posities een taktische waarde toekend (Listing 1).

Listing 1: getMoveScore voor het berekenen van de taktische waarde van een zet

```
private static Integer getMoveScore(String move, TttBoard b)
    {
            int score = 0;

            /* find all lines that hold the given position (xxx - ooo) */
            ArrayList<String> lines = new ArrayList<String>();
            lines = getLines(move, b);

            // Heuristics for valuating given lines
            for (int i=0; i<lines.size(); i++)
            {
                if( win(lines.get(i))) score = score+5;
                if( possible(lines.get(i))) score = score+2;
                if( block(lines.get(i))) score = score+2;
                if( free(lines.get(i))) score = score+1;
            }

            return score;
    }
```

Iedere rij waarin de gegeven positie te vinden is heeft één van de volgende eingenschappen:

1. win(): Door een zet op de gegeven positie wordt de rij compleet voor de computer (drie circels)

2. possible(): Door een zet op de gegeven positie zijn er twee circels en een vrij veld op een rij. De computerspeler zou deze rij dus in een latere zet kunnen winnen

3. block(): Door een zet op de positie wordt een rij van de tegenstander geblockt (twee rondjes en één kruisje)

4. free(): Er zijn nog geen stenen in de rij

Deze verschillende toestanden worden in getMoveScore nu waardes toegekend. Iedere mogelijke positie verzamelt zo een score voor zijn individuële taktische waarde. De positie met de hoogste score wordt dan gezet als beste move.

Implementatie van PathPlanning

De implementatie van het padplanning gedeelte kwam neer op een simpele aanpassing van het PP.java bestand voor het schaakspel. Omdat het pad alleen een variabele kent (de positie van het paaltje waar de steen op moet komen) en ook geen obstakels in de weg kunnen zijn, kon het gehele low-path planning gedeelte worden verwijdert en het high-path gedeelte gereduceerd tot negen stappen (Listing 2).

Listing 2: De negen stappen voor het plaatsen van een steen

```
    fromPoint.z = SAFE_HEIGHT;
    p.add(new GripperPosition(fromPoint, b.theta, OPEN_GRIP));

    fromPoint.z = LOW_HEIGHT;
    p.add(new GripperPosition(fromPoint, b.theta, OPEN_GRIP));

    fromPoint.z = PIECEHEIGHT*0.75;
    p.add(new GripperPosition(fromPoint, b.theta, OPEN_GRIP));

    p.add(new GripperPosition(fromPoint, b.theta, CLOSED_GRIP));

    fromPoint.z = SAFE_HEIGHT;
    p.add(new GripperPosition(fromPoint, b.theta, CLOSED_GRIP));

    toPoint.z = SAFE_HEIGHT;
    p.add(new GripperPosition(toPoint, b.theta, CLOSED_GRIP));

    toPoint.z = LOW_HEIGHT;
    p.add(new GripperPosition(toPoint, b.theta, CLOSED_GRIP));

    p.add(new GripperPosition(toPoint, b.theta, OPEN_GRIP));

    toPoint.z = SAFE_HEIGHT;
    p.add(new GripperPosition(toPoint, b.theta, OPEN_GRIP));
```

Inverse Kinematica

Het inverse kinematicagedeelte kon één op één worden overgenomen van het schaakprogramma omdat hier echter geen veranderingen moeten worden geïmplementeerd.

De GUI

Voor het maken van de GUI is een een stuk code gebruikt van Blmaster (4). Deze code bevatte de implementatie voor het spelen van de originele versie van Tic Tac Toe. Voor het maken van de versie in dit verslag moest het meeste worden aangepast. De GUI maakt gebruik van JFrames, JButtons en andere J-items voor het opzetten van het venster en het speelveld. Er wordt gecontroleerd of de speler een legitieme zet doet (dus als de zet op de onderste laag is óf als er blokjes onder de betreffende plek liggen is de zet legitiem). Dit gebeurt door de voorwaarden te controleren met een if-else statement. Verder is er een methode om xyz coordinaten (bijvoorbeeld 111) om te zetten in het juiste nummer van de JButtons (er zijn 27 JButtons, één voor elk vakje in het speelveld). Op deze manier kan een zet afkomstig van de MoveGenerator worden geplaatst in het juiste vakje van de GUI. Onjuiste zetten zorgen voor een message die dit aanduidt, en als alle vakjes zijn gevuld wordt de winnaar bepaald door de 3 op een rij voorkomens te vergelijken (het tellen van de sequenties wordt constant gedaan). Hierbij wordt ook gevraagd of de speler een nieuw spel wilt beginnen. De volgende sectie laat de GUI zien.

Resultaten

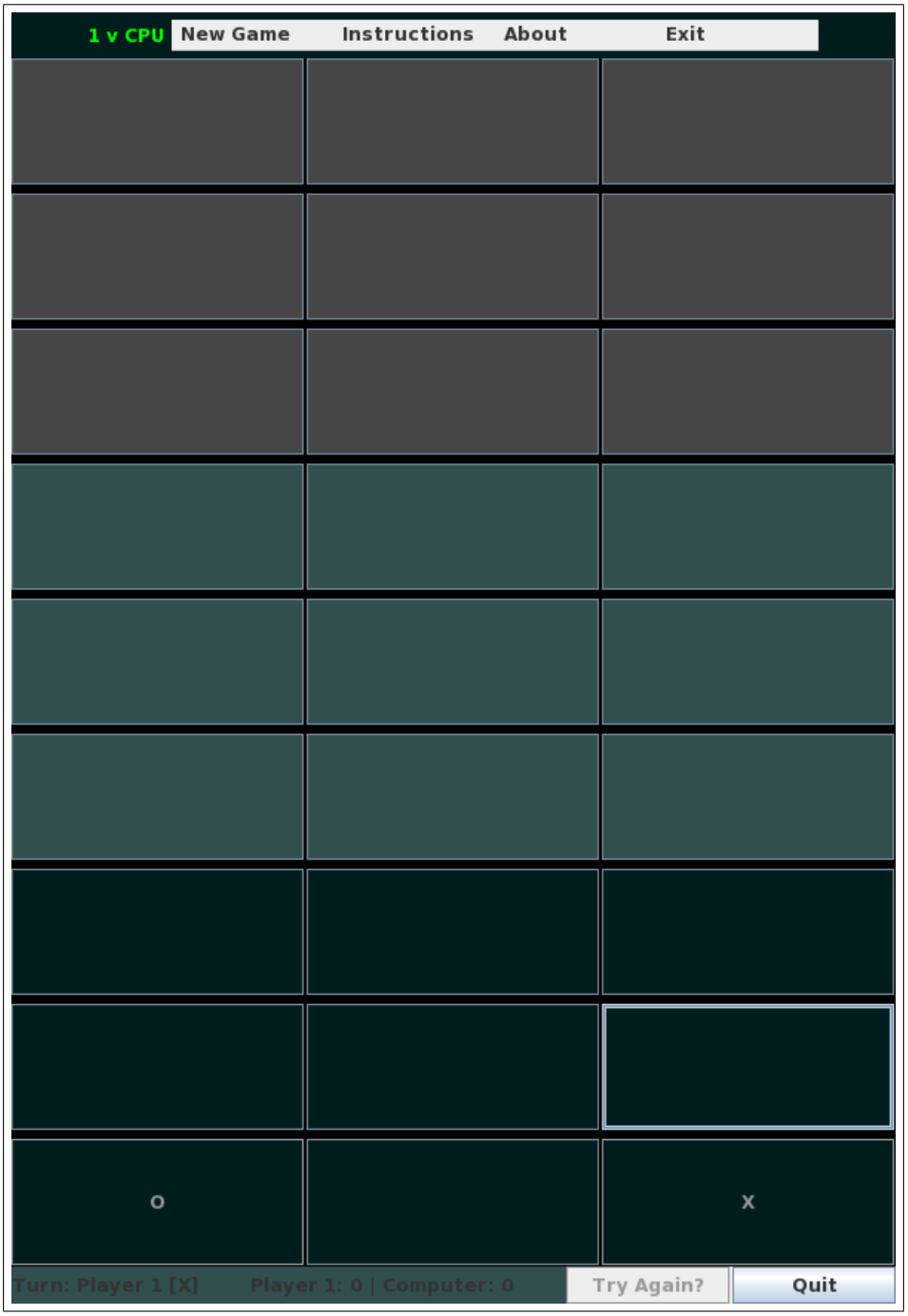
De hierboven beschreven methodes leidden tot een volledig zelfstandige softwareapplicatie die hier zal worden toegelicht.
Het programma wordt opgestart door de command

```
java TicTacToe
```

in de map met alle ontwikkelde bestanden. Hierdoor wordt het hoofdprogramma gestart een de GUI getoond.

De GUI van het mainthread programma

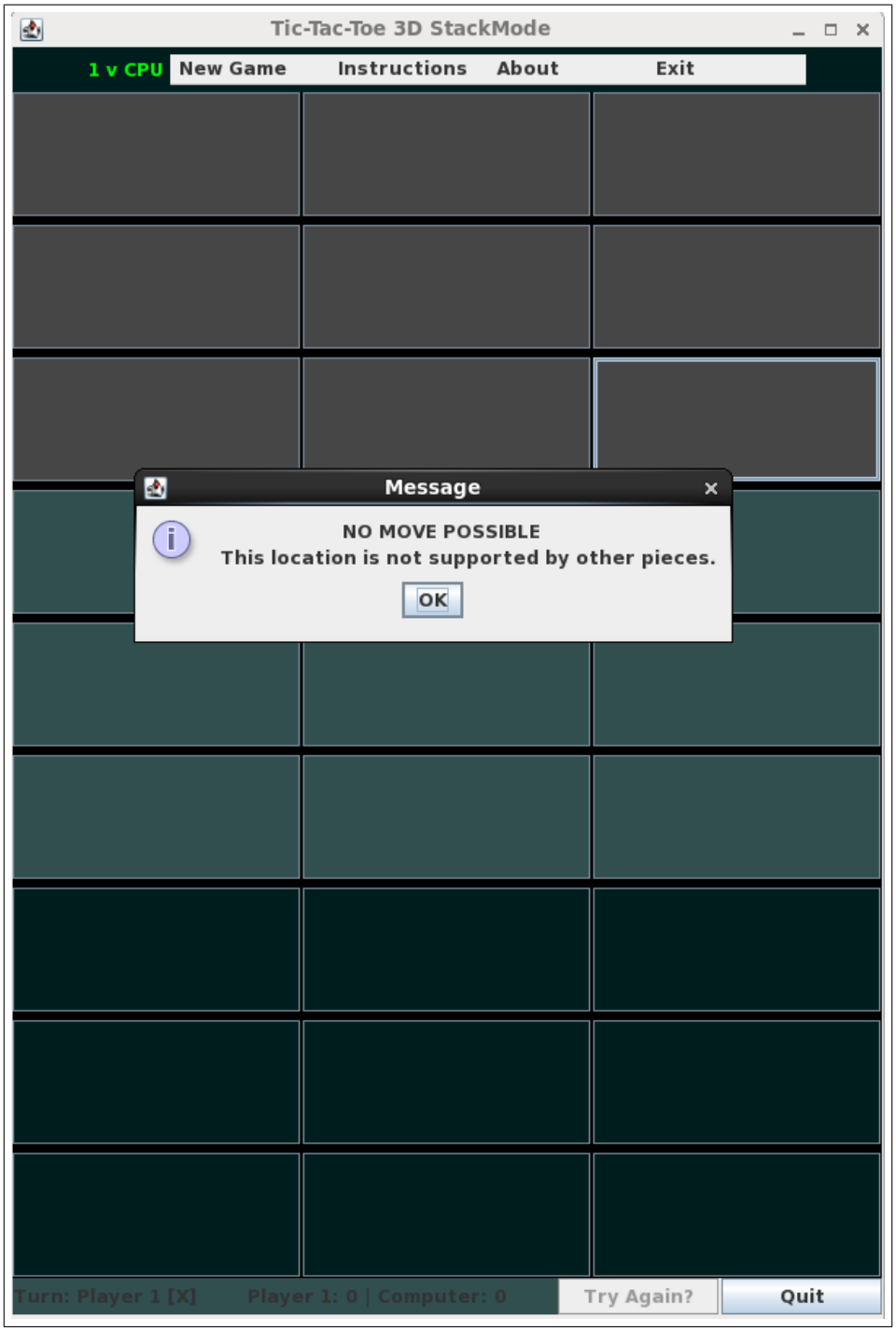
Hier kan vervolgens meer informatie over het programma en de ontwikkelaars worden opgevraagd, of de namen van de spelers worden aangepast. Door een klik op de button "New Game"kan voor een nieuw spel - mens tegen mens of mens tegen computer - worden gekozen. De voor de simulatie interessante keuze is hier dus "Player vs Computer".



De spelrepresenatie van het GUI

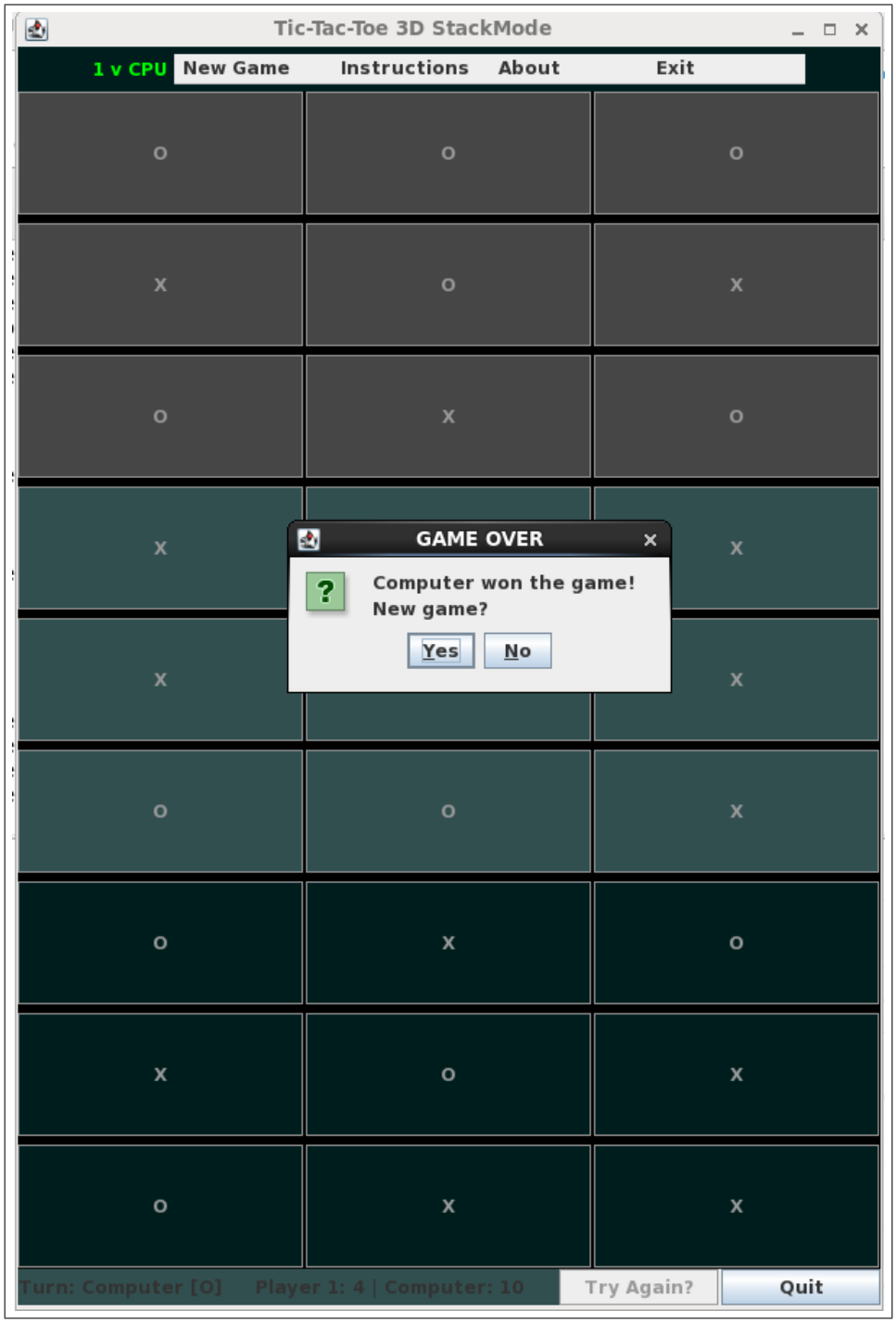
Het spelbord wordt getoond in een eenvoudige 2D representatie met de drie lagen als gewoone Tic-Tac-Toe velden boven elkaar. De menselijke speler mag beginnen en een kruisje in een van de negen velden van de

onderste laag zetten. De computer reageert hierop met zijn als beste bevonden tegenzet. Een klik op een veld uit de tweede of derde laag dat niet ondersteund is door stenen in de ondere lagen wordt niet geaccepteerd en leidt tot een pop-up venster.



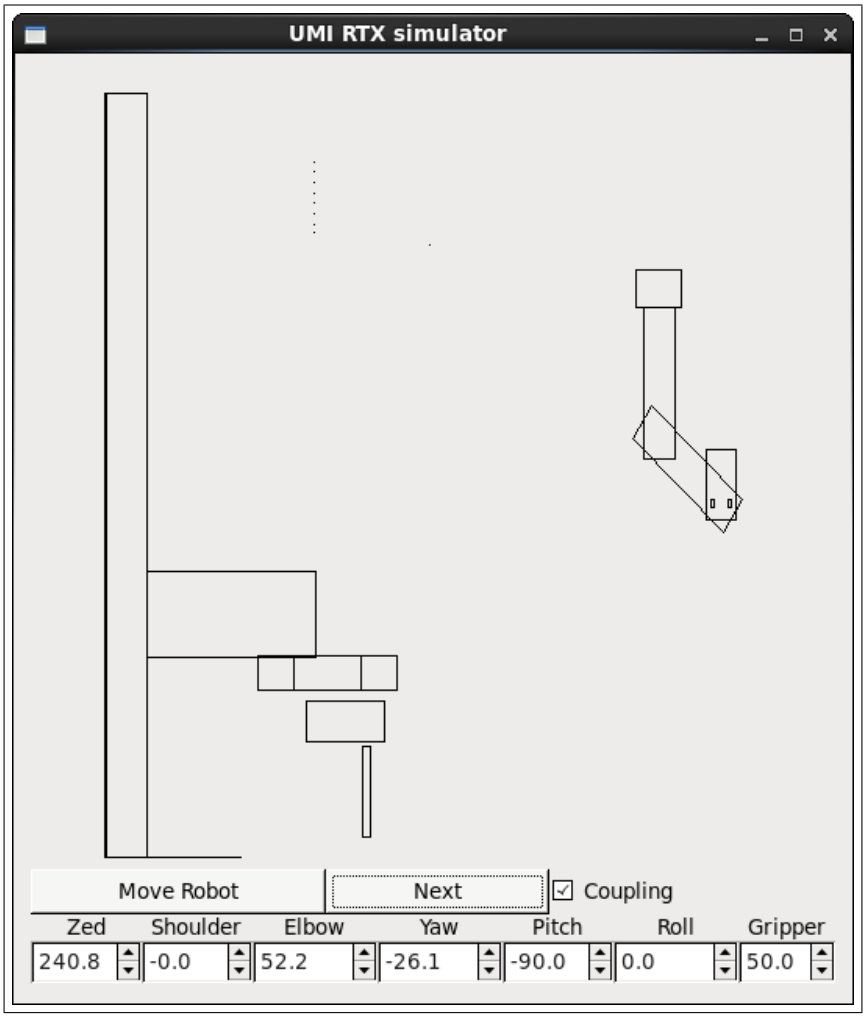
Een pop-up dat waarschuwd voor een verkeerde zet.

Als het spel is afgelopen wordt ook dit als een pop-up getoond en de winner bepaald. De exacte score is aftelezen in de hoek links beneden.



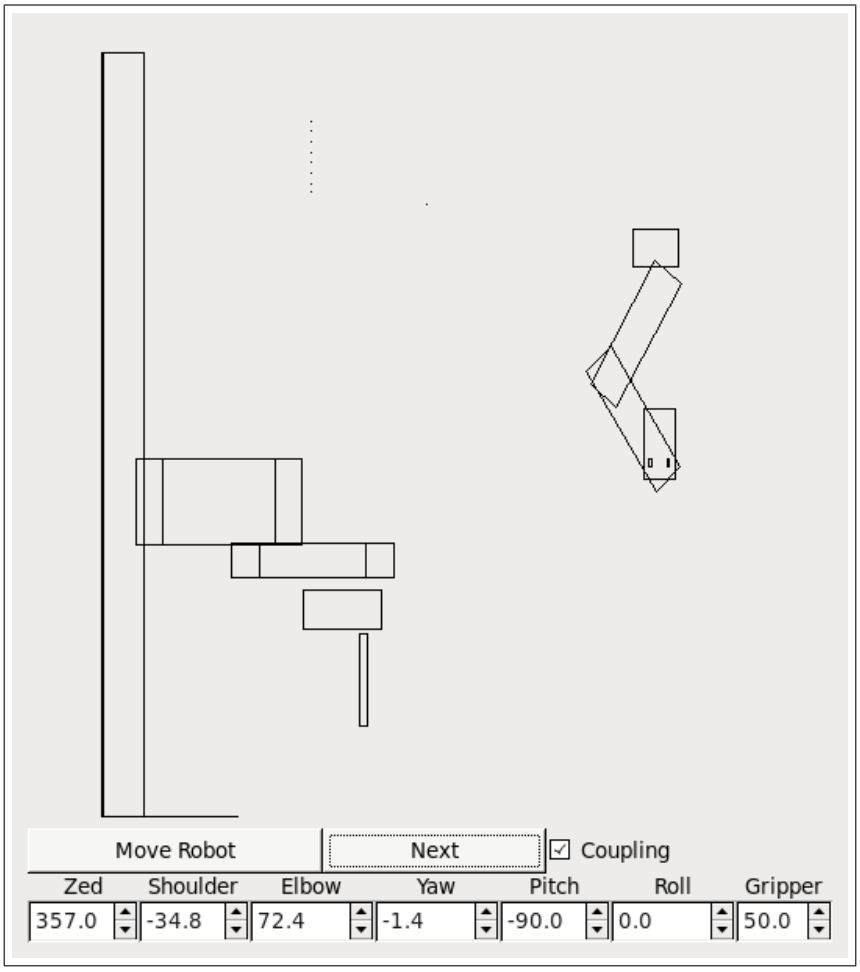
Een volledig uitgespeeld spel met eindscore 10:4 voor de computer. (Dit is echter alleen een voorbeeld, de menselijke speler kan ook duidelijk meer punten scoren)

Iedere computerzet triggert de PP.java en IK.java bestanden die een coordinatenberekening en inverse kinematica transformatie uitvoeren om de bijbehorenende armbegeweging te genereren. Één move doorlopt hierbij een bepaald aantal stappen waarvan hier de kenmerkende worden getoond:
Van de beginconfiguratie van de arm beweegt hij naar een bepaalde positie waar nieuwe stenen verkrijbaar zijn. Deze positie bevindt zich rechts in het midden naast het spelveld op nul hoogte.



De gripper op positie van een nieuwe steen buiten het veld.

Van hier beweegt de arm naar de juiste positie boven het spel waar hij de steen loslaat zodat hij langs de paaltjes op de juiste plek valt.



De gripper op positie boven een paaltje van het spelveld.

Conclusies

Één conslusie is dat het goed mogelijk is nieuwe varianten van een bestaand spel te implementeren met een zelf ontworpen AI. Belangrijk is wel dat de huidige versie nooit getest is op de Umirtx arm zelf. De reden hiervoor waren technische problemen, die niet binnen de tijd door onze superieuren konden worden opgelost. Wel heeft alles gewerkt op de umirtxsimulator, waardoor het in theorie ook zou werken op de echte robot arm.

Een tweede conclusie die uit dit project en het verloop ervan getrokken kan worden is dat het niet altijd baat een AI algoritme zo slim mogelijk te maken. Wanneer men dit niet in acht neemt kan men een applicatie ontwikkelen die veel rekentijd en rekenkracht kost. Ook kan het zijn dat de AI te slim is en dat de menselijke speler het spel nooit meer zou kunnen winnen. Dit neemt elke vorm van spelplezier weg.

Discussie

Nu het project ten einde is kan er worden teruggekeken op het geleverde werk en het behaalde resultaat. Het behaalde resultaat voldoet volkomen aan de verwachtingen en vooraf gestelde eisen. Echter, er zouden nog functies kunnen worden aangepast of toegevoegd om het spel verder te ontwikkelen.
Een functie die in de categorie 'nog te verbeteren' valt is die waar de waarde van een rij wordt bepaald. Dit is de methode genaamd getMoveScore (te zien in Listing 1, blz 6). De waarden die hier worden toegekend

zijn wegens de tijdsdruk slecht enkele malen getest. Wanneer er meerdere testsessies mogelijk waren geweest, hadden de waarden kunnen worden geoptimaliseerd. Het resultaat hiervan zou zijn dat de AI kant van het spel nog iets meer daadwerkelijke intelligentie zal kunnen vertonen.
Een eventuele functie ter uitbreiding van de ontwikkelde applicatie zou een functie kunnen zijn waarmee de gebruiker een moeilijkheidsgraad zou kunnen instellen. Op deze manier kunnen er meerdere slimme en minder slimme algoritmen worden opgesteld en geïmplementeerd. De gebruiker kan vervolgens kiezen tegen welke vorm van de AI hij zou willen spelen. Dit is een toevoeging op de applicatie aangezien het spel hierdoor aantrekkelijk wordt voor mensen van verschillende niveau's.

Referenties

1. http://forum.codecall.net/topic/43209-java-source-code-tic-tac-toe-game/
2. http://www.greendiary.com/wp-content/uploads/2012/07/43d_tic_tac_toe_image_title_upvex.jpg
3. http://zone.msn.com/en/general/article/generalmessengergames.html
4. http://forum.codecall.net/topic/43209-java-source-code-tic-tac-toe-game/