Final Report
AI for Settlers of Catan

Group 22: Hans Bjerkander, Christian Lidberg, Daniel Mauritzson, Johan Olin

May 27, 2011

Abstract

Settlers of Catan is a very popular board game which the whole project group enjoy. This report presents our project where we developed an AI for a bot, using a multi-agent system implemented in Erlang, with the focus on utilizing development cards. The AI has been tested against the build in bots in JSettlers2 and have been able to get high points, and sometimes even win, but it's dependent on luck.

Contents

1 Introduction and Background

1.1 The problem you tried to solve

To create an AI for a bot that could play the board game Settlers of Catan (SoC), see rules [4], using a multi-agent system written in Erlang. The focus was to develop an AI that used development cards in an efficient way. The problem can be split into a number of sub problems:

- Choose start positions that fit the strategy.
- When development cards should be bought.
- Utilizing development cards in a way such that their profit is maximized.
- How to expand: Upgrading settlements, build new ones or/and get a harbor. Because we ran out of time we limited this to upgrading settlements not build new roads and settlements.

1.2 Results from the literature

Saleem and Natiq master's thesis [1] is about creating a multi-agent bot for Settlers of Catan. Their thesis focus on the trading part of the game and compares how different parameters affect the bot. It uses a static order of how development cards should be played, something that we haven't used in this project, but the paper still has a good description of how to implement a Settlers of Catan AI. The paper written by Branca and Johansson [2] is also about creating a Settlers of Catan bot and compare it against other bots. In comparison to that paper, this project implemented a better use of development cards and more strategies regarding the largest army. The weight and utility functions in the paper had good descriptions and we based our functions on them. The paper by Johansson [3] has a good general description on how to implement a multi-agent system[6, ch. 11.4] and how it competes against different AI:s.

1.3 Tools and programs available

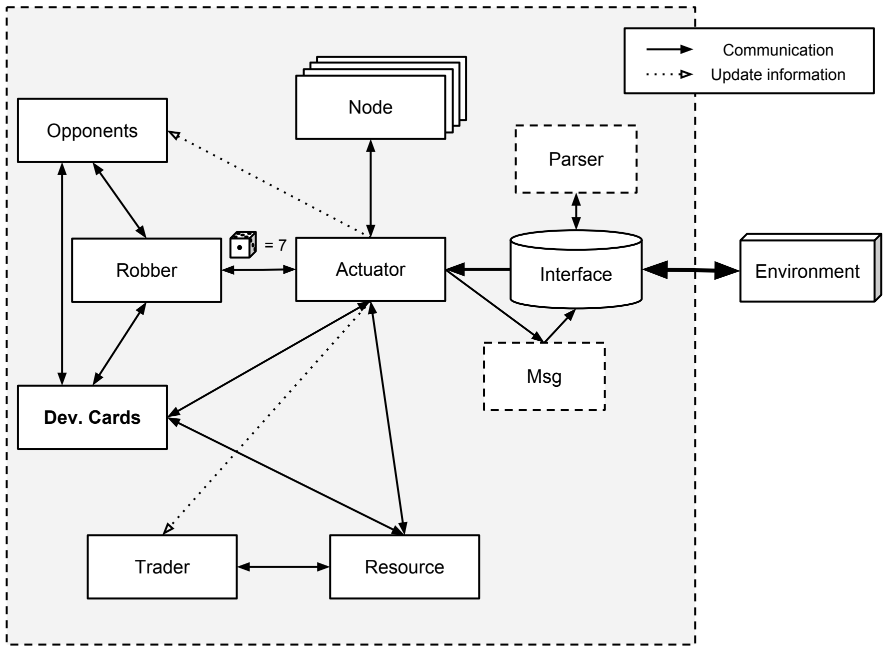
The game server, jSettlers2 [7], has been perfect for this problem. It can be modified so that no human player is needed, and it have a set of bots that can be played against, thus can be used as a benchmark. The communication with the server is done through TCP sockets and thus any programing language that supports this can be used to create a bot. Some difficulties has been encountered when interpreting the messages because the server expect the bot to know what to do in special cases, like the initial phase, and doesn't always send every message. Even though it took some time to learn how the server worked with messages and what everything meant, the jSettlers2 have given us more time to spend on the actual AI. A practical difficulty with jSettlers2 is that there is no way to see how many resources that are available or what resources an opponent discard. That information is important, and is available when playing the actual board game, when deciding whether or not to play a monopoly card.

2 The central idea in your solution

The central idea was to create an AI for SoC using multi-agents implemented in Erlang. Erlang has been a good choice because of its efficient message parsing. The agents cover different parts of the game and together they communicate and make decisions depending on the environment, which is obtained and updated from the jSettlers2 server through the game. Since the agents are independent it was easy to change and optimize a single agent without affecting the others. Also the performance become optimal since the calculations is decentralized to all agents. The goal to create an AI that make good use of development cards was the main subject in this project. Therefore the card builder strategy[5] was used when setting up the game to give the AI the prerequisites needed to make fully use of development cards. This means that ore and grain is prioritized so it can get two fast cities and then concentrate on development cards. In contradiction to the paper[1], this AI e.g. doesn't play development cards in a specific order, neither does it always play a development card just because it has one. Instead it uses a weight function for development cards that depends on the environment.

3 Overview of the architecture

3.1 Running modules



3.2 Running modules

3.2.1 Actuator Agent

The Actuator is the central agent in this project and the one taking the final decisions. It is implemented to the extend that it receives all the messages that is significant for the gameplay, but it doesn't handle them all. The Actuator is able to play the initial phase (put out two houses and two roads), read and initiate the board, roll the dice, reject offers, discard cards, make bank trades, upgrade a settlement to a city, play the development card year of plenty and the knight card , play the robber and choose player to block. The play for the initial phase is triggered by certain game states, but in this phase this isn't enough for the Actuator to know exactly when to play settlements and roads. So it got an internal state that is depending on which seat the AI has. When in the initial state representing our turn, the Actuator calls every Node agent to get information about their dice-probability and resources which is used to choose placement of settlements and road. The algorithm for the initial phase is pretty simple and depends on the dice-probability for every node, but nodes with access to ore, grain and harbors get a larger weight. The placement of the roads isn't implemented so it just puts a road to the right of the settlement. The Actuator also receives the boardlayout at the beginning and sends to every node and to the Robber. It updates the board whenever something is built or the robber is moved. The boardlayout consists of information about resource and dice-probability for every tile. Whenever an offer from another player is received, it will automatically be rejected since this project doesn't implement that. Discard cards, play the robber and choose player to block is calculated in the Resources-, Robber and Opponents agent. The result from the agents isn't returned to the Actuator since these moves is mandatory and doesn't need to be compared to other decisions. The Actuator also receives the resources that the AI get every turn and sends the update to the Resources agent.

Pseudocode for deciding which development card to play

```
ActuatorDecision()
    receive proposals
    knightValue = value from proposal
    check which build suggestions that can be built
    monopolyValue = victory points we can get from building
    yearOfPlentyValue = victory points we can get from building
    RoadBuildingValue = value from proposal
    if only one dev card
      if we only got one knight card
        play knight when value > 1.0
      else if we only got one monopoly card and got 8 or 9 points
        play monopoly when value > 0.5
      else if we only got one monopoly card
        play monopoly when value > 1.5
      else if we only got one year of plenty
        play year of plenty when value > 0.5
      else if we only got one road building card
        play roadbuilding when value > 0.5
    else
      max(knightValue,monopolyValue,yearOfplentyValue,RoadBuildingValue)
      play dev card with max value, when max value > 0.5
    build and buy from the sugestion with the highest value
```

3.2.2 Bot Agent

The Bot agent is responsible for initiation of all agents, sending the agents the process ids they will use for internal communication and starting the game. After this, the Bot agent isn't of any use and its process terminates.

3.2.3 Interface Agent

The Interface Agent connects to the server, parse and forward the messages to the Actuator agent and sends messages from the Actuator to the server. It uses the parser module for parsing messages from the server. When a message is to be sent to the server it goes through the msg module to parse it into a server message.

3.2.4 Node Agent

There exist 54 Node agents, one for every node. It keeps information about adjacent tiles, nodes, its own dice-probability and resources. Whenever a settlement gets built the node updates its status as blocked and also tells the adjacent nodes that a settlement can't be built at their location because of the block caused by this node. It also keep tracks of the roads around the node.

3.2.5 Opponents Agent

The Opponents agent keeps track of the opponents resources, development cards, victory points and who has the largest road or/and army. When the AI moves the robber, this agent decides who to take a resource from. The player that's blocked and has the highest score, is chosen. If fully implemented, the agent would calculate which card it mostly wants and pick from the player with the highest probability to hold that card. The part where it keeps track of the opponent resources is also not implemented.

3.2.6 Resources Agent

The Resources agent keeps track of the AI:s resources and discards the number of resources required when the dice result is seven and the AI got more than seven cards. Since the card builder strategy is used clay and wood are discarded first, and if it need to discard more cards it iterates over the rest and pick one at a time so it will discard evenly over them. The agent also, in cooperation with the Trader agent, calculates which trades to make with the bank depending on what it can build. This is done by getting all the possible hands that can be traded for, including the original hand, from the trader. Then it calculates what each of the hands can build and then it put weight on them and picks the best one. Since the AI can't expand with new settlements only cities and development cards will increase the weight of the hand. Cities are worth more than development cards since a city will increase the amount of cards received and thus will be more profitable in the long run. It also uses some plan ahead, if the AI almost can build a city, i.e. it misses one resource, it will not buy any development card but instead wait for the city.

3.2.7 Robber Agent

When asked, it calculates the best robber placement and sends the result to the server through the Interface agent. The spot is chosen by comparing all tiles, the tile with most settlement and cities with the same owner is chosen. This is done by calling adjacent Node agents to the tile and with that information weight every tile to choose the best one to block. Tiles that the AI have adjacent settlements or cities will not be blocked.

Pseudocode for moving the robber

```
for all tiles
  if not desert tile then
    get adjacent nodes
    if we dont have settlement at adjacent nodes
      for each node
        if settlement
          increase num_settlements with 1
        else if city
          increase num_settlements with 2
      num_different = number of players around that tile
    else
      check next tile
  else
    check next tile
  robber_scale = num_settlements/6
  if num_different > 0
    robberscale += 1 - num_different/4
return tile with highest robberscale
```

3.2.8 Trader Agent

The Trader agent keeps track of all usable harbors and given a resource hand, it calculates all possible hands with the use of trades. Then the result is returned to the Resources agent, which will calculate what's possible to build with them. It is also used when calculating all the possible hands that can be traded for when a year of plenty is played for the DevCard agent.

3.2.9 Devcard Agent

The DevCard agent keeps track of the development cards held by the AI and weight the value of playing them considering the environment. One thing that is taken in consideration is the game phase which is divided into early-, mid- and late game. It is dependent on the amount of victory points that the leader has, 1-4 points is early game, 5-7 points is mid game and 8-9 points is late game. At the beginning of every turn, the Actuator requests the values for every specific development card. The values is sent to the Actuator which decides which cards to play.

Knight Cards The decision whether or not to play knight card is mostly dependent on the current game phase. Early game the AI avoids playing a knight card unless it's blocked or an opponent is going for an early largest army. The type of resource that is blocked is also considered, since the strategy used doesn't need clay or wood the value of playing a knight is lowered because it often worth saving the knight for when a needed resource is blocked. When the game phase is mid- or late game it starts to play more knights to get and defend the largest army. Also at this time of the game, blocking other players is more beneficial because they probably have expanded with cities and settlements and will be denied more cards than early game.

Year of Plenty Cards The year of plenty cards is played when two extra resources can contribute to a build or buy. This card can be very helpful for upgrading a settlement to a city if the AI doesn't have high probability spots for both ore and grain. The DevCard agent asks the resources agent what builds that can be done using this card and sends the received proposal to the Actuator.

Monopoly Cards The monopoly cards should work similar to the playing of year of plenty but since there is no way to get exact information about how many cards of a specific type that is in play, it can't fully utilize this card.

Road Building Cards The AI never plays a road building card since the expansion of roads and settlements isn't implemented so it wouldn't be of any use. But when that is done, this card will help the AI a lot since it doesn't get much wood and clay because of its strategy. By getting two free roads it can expand to two new nodes and if a settlement is built, get more resources to use. This card also opens a new way to get victory points, namely by the longest road. That is though hard to get because the AI won't build many more roads due to the lack of clay and wood. The weight of playing a road building card is pretty simple at the moment and not dependent on many things. The game phase shouldn't be considered here since expansion of roads and settlements is important all game.

Pseudocode for playing development cards

```
if we got knight card
  if the robber blocks our node/nodes
    movevalue = call moveRobberVal(1)
    send proposal(play knight,movevalue)
  if we can get largest army
    movevalue = call moveRobberVal(2)
    send proposal(play knight,movevalue)
  if we need to def largest army
      movevalue = call moveRobberVal(3)
    send proposal(play knight,movevalue)
if we got monopoly card
  for each resource type, r
```

```
    get how many cards we will get if we play monopoly
    send cards to resources
    receive all possible actions we can do
  {re,action} <- fetch r with highest action value
  send proposal(play monopoly, re, action)
if we got year of plenty card
  send all possible pair of resource cards, (r1,r2) to resources
  receive all possible actions we can do
  for each pair
    send proposal(play year of plenty, (r1,r2), [all possible actions]
if we got road building card
  if we can get longest road
    if we can win
      return 5.0
    else
      return 2.0
  else
    return 1.0
  send proposal(play road building,value)
```

Pseudocode for moveRobberVal

Notes: Early-, mid- and late game isnt fully defined but will be dependent on the amount of victory points the leader has. For our strategy, valuable resources is ore, wheat and sheep.

```
moveRobberVal(case)
  if case == 1
    if opponent got 8 or 9 points
      return 1.5
    else if (mid game or late game) and blocked resource is valuable
      return 1.5
    else if (mid game or late game) and blocked resource isnt valuable
        return 0.5
    else if early game
      return 1.0
  if case == 2
    if early game
      return 0.0
    else if we got 8 or 9 points
      return 5.0
    else
      return 1.0
  if case == 3
    if early game
      return 1.0
    else
      return 1.5
```

3.3 Modules designed but not implemented

3.3.1 Devcard Agent

Monopoly card isn't utilized since there is no way the get information about how many cards of a resource type is in play. So to be able to implement this the resources in play need to be estimated so a monopoly card isn't played when the resource gain is very low. Road building isn't utilized either since the AI can't expand, so there is no need for building road.

3.3.2 Node Agent

The Node agent doesn't handle roads at the moment. This should be implemented so it can communicate with other Node agents, not just the adjacent Node agents, and calculate which nodes the AI can build to and the value of that action. With that information the Actuator can make decisions regarding which nodes that are worth building towards.

3.3.3 Resources Agent

The discard function in the Resource agent could use some more optimization and take possible builds into considerations. This is essential if the ability to expand is added, making clay and wood useful. Also the weight function needs to be updated because the current one uses weights only on development cards and on cities. The plan was to implement it using the forumulas below but due to lack of time it wasn't done. The formulas are based on the paper [2] but slightly modified to support a more dynamic behaviour.

The Resource agent will calculate all possible actions doable with the current resources, with or without trade, and send the actions to the Actuator agent. There's 4 different actions that cost resources: buy a development card(A_d), build a road(A_r), build a settlement (A_s) or build a city(A_c). All actions will give a value that's based on how many victory points or resources they can get divided on the cost of that action. In the calculation of building a road (A_r) it also calculates what new action it can yield and therefore a road can give a high value even if it don't give any direct victory points.

$$\begin{aligned}
A_d &= \sum_{n \in cards} \frac{W_n}{|Cards_n| * C(n)} \\
A_s &= \frac{VP(s) + U_t}{C(s)} \\
A_c &= \frac{VP(c) + U_t}{C(c)} \\
A_r &= max(\frac{VP(r)}{C(r)}, \frac{VP(r)}{C(r)} + max(A_r, A_s))
\end{aligned}$$

Weight function for a node:

$$U_t = \sum_{t \in neighbourTiles(n)} WeightResource(t) * Probability(t)$$

Collectible resource factor

$$H_r = \begin{cases} 2 & \text{if the bot don't have settlement/city with acess to resource r} \\ 1 & \text{else} \end{cases}$$

Cost function

$$C(r) = \begin{cases} 3{*}W_g{+}2{*}W_o & \text{if } r = \text{City} \\ W_b{+}W_l{+}W_g{+}W_w & \text{if } r = \text{Settlement} \\ W_b{+}W_l & \text{if } r = \text{Road} \\ W_g{+}W_o{+}W_w & \text{if } r = \text{Develompent Card} \end{cases}$$

Victory point function

$$VP(r) = \begin{cases} 2 & \text{if } r = \text{City} \\ 1 & \text{if } r = \text{settlement} \\ 2 & \text{if } r = \text{road and it gives us the longest road} \end{cases}$$

List of weights used:

Parameter	Value	Explanation
W_b	max(1,1.2-T*0.02)	Brick weight
W_l	max(1 , 1.2-T*0.02)	Lumber weight
W_g	min(1.2, 1+T*0.02)	Grain weight
W_o	min(1.2, 1+T*0.02)	Ore weight
W_w	1	Wool weight
$W_{h2:R}$	max(1, 0.6*NrOfTiles(R))	2:1 Harbor of resource R
W_{h3}	max(0.8, 0.5*$max_{r \in R}$(NrOfTiles(r))	3:1 Harbor
W_v	1	Weight for Victory Point card
W_m	2.5	Weight for Monopoly card
W_r	1	Weight for Road building card
W_k	0.5	Weight for Knight card
W_y	1	Weight for Year of plenty

T is which turn the game is in, so brick and lumbers weight will decrease and grain and ores weight will increase, the longer the game plays out. This is because grain and ore is needed to upgrade settlements to cities, which is important step in the later part of the game, and brick and lumber is most useful in the beginning of the game.

3.3.4 Opponents Agent

The Opponents agent is mostly done but doesn't keep track of the opponents resources at the moment. Since the server tells the Actuator when a player get resources, this wouldn't be hard to implement. The difficulty in this part is when an opponent looses cards due to robber or discarding because of a 7. Then the server don't tell the AI which cards were drawn or discarded so the Opponents agent need to calculate probabilities for certain cards. This is used when the AI should take a card from an opponent due to movement of the robber.

3.4 Modules a future continuation may have

3.4.1 Expanding to new settlements

One thing that the AI is lacking at the moment is the ability to expand, that is building roads and settlements. Build roads and settlements don't just add victory points but may also provide the ability to get new resources as well as blocking opponents from getting new nodes or the longest road. So adding this would increase the chances of winning as it broadens the possible actions the bot can take. However this is not that easy to implement as there are lots of factors to take into consideration when expanding, but it can be done in several steps.

The first step should be to utilize the starting roads as the point to build from, this means that only one road have to be built in order to be able to build a new settlement, unless it is blocked by an opponent. Here it would be good to increase the value for 3:1 harbors and any 2:1 harbor that we have tiles for, as cheaper trades are always good.

3.4.2 Trading with opponents

Trading is a good way to get resources that you are missing, at the moment the AI is only able to trade with the bank. But the game also allow for trade between players and this opens up the possibility to get cheaper trades and even trades that you can't make with the bank. So implementing this ability in the AI will of course increase the chances of winning. But just as there are numerous things to take into consideration when expanding, trading with opponents is not as straight forward as trading with the bank. Here the opponents gain of the trade have to be considered and to do this knowledge about the opponents current status have to be kept and evaluated to see if the AI get more out of the trade then the opponent.

3.4.3 Support for 6 player games

There exists an expansion for SoC that allows for 6 players, it also adds an extra building phase after every players turn. This phase is played clockwise and each player have the option to build or buy, but only with the cards they have on hand no trading is allowed. JSettlers support this expansion so this could be implemented into the AI. To fully utilize the extra trading phase it's important to have more evenly distributed probabilities for the resources since no trading is allowed in that phase. Also it becomes more important to use some plan ahead so the AI doesn't always buy the cheapest thing whenever possible but instead decide what it want to buy in the near future and wait for that.

4 Results

The AI plays the initial state with good results considering the used strategy. It chooses spots prioritizing both high probability and the needed resources, making it very likely that it will get ore, grain and sheep. If it doesn't get all three resources on the first settlement, those resources gets a higher value when placing the second settlement. The harbors is also taken in consideration when placing the settlements. If the AI can get good use of a harbor and still get good resources at that node it builds there and is able to utilize 3-1 and 2-1 trades. When placing the roads to the initial settlements, they are always put to the right of the settlement, not considering possible future building spots since the AI can't build roads or settlements.The discarding of cards is pretty fair and suits the strategy since it throw away clay and wood first and then starts discarding ore, sheep and grain. The placement of the robber is pretty good and calculates which tile that is best to block depending on how many settlements/cities that is present at that tile. When upgrading from settlement to city almost the same method is used as when placing the initial settlements except it doesn't care about harbors. The AI always buys development cards if possible except when it soon can build a city, then it waits a round to hopefully get the needed resources. This way it gets many development cards to use during the game and often early cities so it collect a lot of resources. The downside is that the AI often need to discard cards since it can't build anything else but development cards when it got two fully upgraded cities.

4.1 Evaluation

The AI is able to achieve good performance, and sometimes even win the game. But it is very dependent on the board layout and how the opponents place their settlements in the initial phase of the game. If the AI can't get all the resources for buying development cards it performs bad. But if it's lucky and gets god spots with all the needed resources it almost always end up with a high number of victory points which show that playing development cards is an important aspect of the game. But to make the AI less dependent on luck the ability to expand with roads and settlements is a must. Trading with the opponents is desirable since that can enable the AI to build faster.

5 References

References

[1]
1. Saleem, H. and Natiq, R.R. (2008). A Multi-agent player for Settlers of Catan , Masters thesis, http://www.bth.se/fou/cuppsats.nsf/all/5182dee54d5efc33c1257559004f8dbc/$file/SOC_Final_Report.pdf
2. The paper is about developing a Multi-agent bot for Settlers of Catan. The bot is focused on the trading and uses develop cards without any strategy.
3. The most important reference to this paper in our document is in Section 1.2 and 2

[2]
1. Branca, L. and Johansson, S.J. (2007). Using Multi-agent System Technologies in Settlers of Catan Bots , Conference Paper, `http://www.bth.se/fou/forskinfo.nsf/alfs/c3607f15e9248a9cc12572eb0053320b/$file/BrancaJohanssonABSHLE07CR.pdf`
2. This paper is about developing a multi-agent Settlers of Catan bot and comparing it against two other monolithic bots. The papers bot was bad but the paper contains a good description of the implementation of a multi-agent system and what to improve.
3. The most important reference to this paper i our document is in Section 1.2 and 3.3.3

[3]
1. Johansson, S.J. (2006). On using Multiagent Systems in Playing Board Games , Conference Paper, `http://portal.acm.org/citation.cfm?id=1160737`
2. This paper is about using a multi-agent bots in boardgames and compares bots in the games Diplomacy and Risk.
3. The most important reference to this paper i our document is in Section 1.2

[4]
1. Klaus Teuber (2007), Settlers of Catan: Game Rules, `http://www.catan.com/en/download/?SoC_rv_Rules_091907.pdf`
2. The official rule book for Settlers of Catan
3. The most important reference to this paper i our document is in Section 1.1

[5]
1. Scott MacPherson (1999), Settlers of Catan Strategy and Tactics Guide, Tactics Guide, `http://files.meetup.com/1550090/Settlers%20of%20Catan%20Strategy%20and%20Tactics%20Guide.pdf`

2. A thorough guide on different strategies and phases the game goes through.
3. The most important reference to this paper i our document is in Section 2

[6] 1. Stuart Russell and Peter Norvig (2010), Artificial Intelligence A Modern Approach, Pearson Education
2. The course book
3. The most important reference to this paper i our document is in Section 1.2

[7] 1. Jeremy D. Monin ‚jSettlers2, webpage, `http://sourceforge.net/projects/jsettlers2/`
2. The projectpage of jSettlers2
3. The most important reference to this paper i our document is in Section 1.3