Softwarepraktikum SS 2016

Assignment 2

Group - 3

Eric Wagner	344137	eric.michel.wagner@rwth-aachen.de
Stefan Rehbold	344768	Stefan.Rehbold@rwth-aachen.de
Sonja Skiba	331588	sonja.skiba@rwth-aachen.de

Task 1

For the first task of this weeks assignment we did write the code to connect our AI to a game server. Therefore we had to write the code that connects to an ip address and the port on which the server is listening.

During the setup phase a lot can go wrong, like not having the right ip or port or trying to connect to an offline server. To avoid any complications, we did try to catch every possible error and output a meaningful error message. After an unsuccessful connection attempt, our program will end itself as a controlled ending is better running under an undefined behaviour. And that way we can make sure that no memory leaks or open ports remain after the end of the program.

After the connection has been set up successfully we retrieve our player number and the map. In the following part of the code, we try to separate the receiving of a message and the control of the game flow as much as possible. To archive this, we did create a **listenToSever()** method which will wait for a message of the server and set a variable to whatever message has been sent. After that it will decode and save the transferred data.
In our main method we will take care of the game flow. First we ask which message was send last and then we execute the according code by using the data saved by the **listenToServer()** method. After a message has been processed, the AI is ready to get receive the next message. This will continue until the end of the game is announced by the server or an error accrued during the data transfer, to which our program would react with an controlled shutdown.

Task 2

We implemented the evaluation method of the bombing phase.
If there are no bombs left, it uses the following functions:

$$Ranking(player) = \begin{cases} MAX & \text{if } Rank = 1 \\ MIN + |\text{Players}| - Rank(payer) & \text{else} \end{cases}$$

Rank is function that returns the ranking of the player, where 1 is first place, 2 the second place and so on. |Players| is the total number of players, that aren't yet disqualified.
The reason why the function is not $Ranking(player) = MAX - Rank(player)$ is, that the search tree possible hasn't the same depth everywhere and may even start in the first phase and continue into the second phase. If the last place would be higher then an average rank from the function, that evaluates the board when the game isn't over, then the algorithmen that uses the search tree will try to achieve that finishing position no matter how good or bad it may be.

Otherwise the game isn't over yet. If that is the case we use the following function to evaluate the board of the player:
$Ranking(player) = \frac{|player\ stones|^x}{|enemy\ bombs|^y * explosionradius^z * Rank(player) * clustering}$
x, y and z are weights that will be changed on a regular basis to make the evaluation as good as possible. *clustering* is the number of cells of the player that are adjacent to each other.
This formula may change if a better one will be found or thought of in later stages of the programming.

Now to the question if the method is time efficient. To be able to compare better, we first make the following assumptions:

- Bomb evaluation method only gets called in phase two of the game.
- The whole board are only valid stones at the beginning of phase two.
- All cells are captured by not disqualified players.
- The operations checking and increasing, checking and decreasing, copying and counting take the same amount of time.

Here are the two possibilities that seemed most likely:

1. Count the number of cells that each player has at the beginning of phase.
2. Count the cells each time again.

First let us look at the things each of those methods has to do and therefore can get ignored.

- Check if there are any bombs left to check if the first ranking function should be used.
- Sort the number of cell each player has to get the rank of the player.
- Half of the tree are leaves.
- Out of simplicity: The number of bombs gets counted each method call again.

Now let us look at the differences:

Method 1	**Method 2**
• Count the number of cells of each player at the beginning of phase two. • Deduct the number of destroyed cells from the according players when a bomb is dropped. • The number of bombs of each player gets copied with each board copy.	• Count the number of cells of each player at each evaluation call. • Only has to count when the functions gets called in a leaf.

The number of operations of the methods accumulate to the following terms:

1. NumCells + NumBombs * $(2 * \text{RadiusBomb} + 1)^2$
   + NumPlayers * NumCopies
2. NumCells * NumLeaves

Note that the NumCells stays the same the whole game, because they only get marked as destroyed.
$\prod_{j=0}^{\text{NumBombs}} NumCells - j * (2 * RadiusBomb + 1)$ where
$0 < NumCells - j * (2 * RadiusBomb + 1)$ calculates most of the possible moves but obviously not all. If you assume that one percent of those are checked and half of the checked ones are leaves you get the following:

$NumOfCopies = [\prod_{j=0}^{\text{NumBombs}} NumCells - j * (2 * RadiusBomb + 1)] \div 100$ where $0 < NumCells - j * (2 * RadiusBomb + 1)$

$NumLeaves = NumCopies \div 2$

If a board is 900 cells big, about 18 bombs destroy most of that board and are the boarder for j in the calculation of NumOfCopies. So NumCells = 900 and NumCopies $\approx 10^{46}$. Thus NumLeaves $\approx 5 * 10^{45}$. If you look at how big those numbers are, you can see the potential irrelevance of NumCells + NumBombs * (2 * RadiusBomb + 1)^2. If you calculate the number of operations with 6 players you get the following:

$$\text{Method 1} \approx 6 * 10^{46} \qquad \text{Method 2} \approx 4,5 * 10^{48}$$

In other words method 1 saves a lot of operations, so why do we use method 2 instead of method 1?
At this point it is not clear if switching to method 1 is worth it, because we would have to change mid game, when the first time the bomb evaluation gets called, into a second class or extend the current class that contains the map information with the number of cells that remain. Using the same class for phase 1 and 2 would add copy operations for each copy in phase 1 also, which would raise the number of operations for method 1 above method 2. An advantage of splitting it into a second class would be that we could remove the information if a cell got deleted out of the phase 1 class that will be currently be copied in every copy of our map in phase 1.