Machine Learning Project (5EC)

Reinforcement Learning for "Dicewars"

1 Introduction

In this project you will build an artificial intelligence (AI) that can beat a human player at a board game, either by being better or by exertion. You will become acquainted with the ML field called "Reinforcement Learning". This means that you will not start with a data set containing features and labels on which you train, but rather you will have to perform actions in the *game* and obtain *rewards*. The next game you play, your ML model will make different *actions* based on the previously played games and collect the corresponding rewards. It is the ML task to maximize the rewards by predicting the appropriate actions given the current *game state*. You will have to study and read-in on this topic by yourself, apply it to solve the game and report your findings.
The project can be made by a team of max. 2 students.

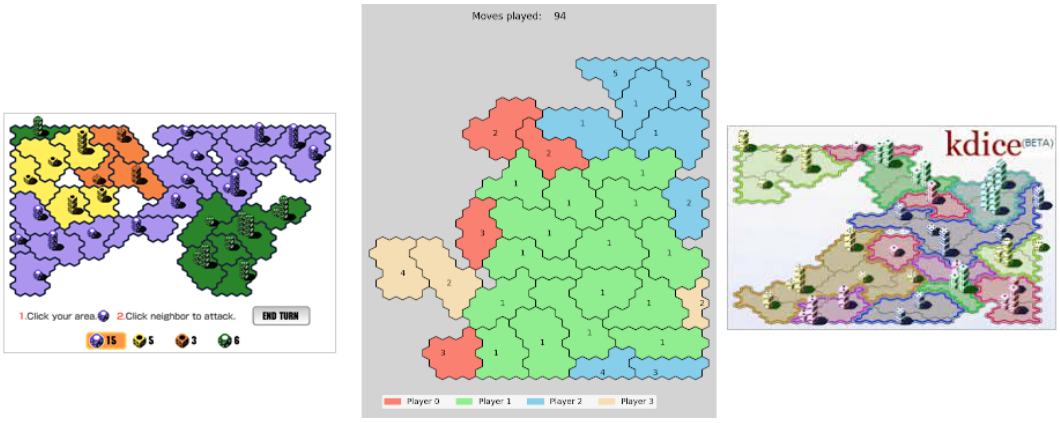

Figure 1: The game *Dicewars*. From left to right: Original Flash game by Taro Ito', Snapshot of the game engine used in the competition, Online multiplayer version called 'Kdice'.

2 Dicewars

If you are not yet already, make yourself familiar with the game (online).
In short, the rules that apply to (**our version of**) the Dicewars game are the following:

1. The game is played with 2 to 4 players.
2. The aim is to take control of all fields on the board. If this state is reached the remaining player is the winner, the others have lost.
3. When it is your turn you can attack adjacent fields owned by another player if you have 2 or more dice on your field.
4. In a battle, all dice of the attacker's field and all on the defending field are casts. The highest sum wins, and in case of a tie, the defender has the advantage.
5. Dice are taken from and moved across the fields based on the outcome of the dice roll. Your AI will learn the rules by experience.
6. The AI will learn how often it can play in its turn.
7. The AI can end its turn at any time.

3 The project tasks

We have adapted a python script for playing "Dicewars" very similar to the openAIgym environments (such as the mountain-car environment). What the code does is to create a python interface that allows you to play the game 'turn-based'. When you pass the action (a tuple of two integers representing the FROM and the TO field, or None) to the code it returns:

```
grid, state = match.step(action),
```

ie. the result of that action. The object `grid` holds the map/board with the fields, and `state` the game state. With `match.render()` you can graphically present the current game observations. The game ends `if state.winner != -1`. A more detailed description with some sample code is given at the end of this document.
The task is in principle very simple, write a function $player$ that makes the following map,

$$player : grid, state \rightarrow action, \tag{1}$$

which maximizes the chance the `state.winner` is your player number.

We have defined the following plan that you and your group should follow.

1. Study the topic of reinforcement learning, there are loads of good websites, books, papers and Youtube videos on the topic. The notebook on reinforcement learning is obviously a good starting point.
2. Think about how your team will build a computer AI for this game. What ML algorithm will you apply? How will this maximize the reward? **Before you begin coding, make a plan on paper in which you sketch your idea! A good approach is to start with a small model (few parameters) and systematically increase its complexity.**
3. Split the the project up in tasks and divide them over the team members. Make sure that all of you contribute to the ML code development.
4. For the ML models you should use the TensorFlow/Keras package (Keras is a high-end interface to tensorflow). Install the required packages (Keras, see instruction below). It is probably wise to make a new python environment for this.
5. Code, test, train and optimize your hyperparameters.
6. **Enter your AI player in the competition against the other AI's.**
7. Write your final project report and upload it on Canvas.

4 Decomposition of the problem

Here we list some aspects of the problem that you could help you find your solution. You might not have time to do all of them, but try and get as far as possible within the timeframe of this project.

1. Analyse the input (`action`) and output class variables (`grid, state`). What are the shapes and types of the avariable that they contain? See also end of this document.
2. Think about how to give rewards to a current observation state. The game does not provide a reward, you have to come up with something yourself.
3. The environment provides an observation of the current game state. You are free to copy this state and adapt the copy (e.g. by extracting only essential information) before passing it to your agent (ML model).

5 Deliverables

- A working python code that learns to play "Dicewars" by reinforcement learning.
- An AI player to play "Dicewars". This should be a function that accepts the current game state and returns an action. The function has the specifications as presented in Section 7. It is important to keep to this structure in order to do the contest between players of different groups.
- A project report containing an introduction to the applied reinforcement learning method, and explanation of the code structure based on the high level routines. Furthermore, report on how you optimized your player, show your player scores versus the number of trained games. Conclude overall evaluation of the player you created, what makes it good or bad, and where and how could it still be improved.

Deadlines:
Submission of your model for the contest: 7-04-2025 before 22:00h.
Hand in final report project: 17-04-2024 before 22:00h.

6 Assessment

Aspects of assessment:

- Understanding of the RL algorithm. Does the group clearly explain and understand the concept behind the RL algorithm that was applied.
- Working of the RL algorithm. Does the group deliver a script/algorithm that clearly shows that the agent learns (by interacting with the environment).
- Tuning the RL algorithm. Does the group clearly describe how they tuned the algorithm (i.e. how and why information from the state is passed to the agent; how and why are rewards tuned; how and why are hyperparameters tuned).
- Interpretation and discussion of the results. The group should reflect on their result. Why did it work/not work; what could be improved and why.

To asses these aspects we will use the following products: report, scripts, contest. Make sure that your code is commented, readable and usable for an outsider. Describe the main layout of your code by means of the high level functions/routines in your project report.

Setup the environment

7 The Player

The AI player should be a class named `Player`. This player class should have a `get_attack_areas(self, grid, state)` method which accepts the current game state and returns an action. The class should look as follows:

```
class Player:
  def get_attack_areas(self, grid, state):
    # YOUR CODE GOES HERE
    action = ...
    return action
```

A description of `action`, `grid` and `state` is given below.

8 The Dicewars environment

The created dicewars environment is available for download on the class's canvas.

On Canvas you will find:

- The dicewars environment
- A working example of the game has also been provided in the `basic_dicewars.py`.

The environment is based on a backend created by Scotty007. Below we will describe the methods, functions, and variables for running the dicewars games. The documentation of the original backend can also be found here. Note that we have modified the source code so the original documentation is not complete, and may not be completely correct.

- **`dicewars.grid.Grid(grid_width=28, grid_height=32, max_num_areas=30, min_area_size=5))`**. This function creates and returns a new grid instance. The grid instance holds the layout of the board. This grid instance does not hold any information over which player owns what area or how many dice are on each area. Example call: grid = `dicewars.grid.Grid()`.
  Arguments:
  - `grid_width`: integer. The width of the board in hexagons.
  - `grid_height`: integer. The width of the board in hexagons.
  - `max_num_areas`: integer. the maximum amount of areas (countries) that will be generated.
  - `min_area_size`: integer. the minimum amount of areas per hexagon.
- **`dicewars.game.Game(grid=None, num_seats=4)`**. This function creates and returns a new game instance. The game instance creates the starting configuration of the match by dividing all the areas in the grid among the players and dividing the dice over the areas. Example call: game = `dicewars.game.Game()`.
  Arguments:
  - `grid`: grid object. The grid which will be played on. If `None` a new grid instance will be created.
  - `num_seats`: integer. The number of players in the game.
- **`dicewars.match.Match(game=None)`**. This function creates and returns a new match instance from a game starting configuration. Example call: match = `dicewars.match.Match()`.
  Arguments:
  - game: game object. The initial configuration of the match.

  The match, game and grid objects can be stored to a file or loaded from a file by using the following methods.
  - `match.save("filename.pkl")`: Saves the current state of the match to the file `filename.pkl`.
  - `dicewars.match.Match().load("filename.pkl")`: Loads a match from the file `""filename.pkl"`. Example call: `match = dicewars.match.Match().load("filename.pkl")`

  These methods work similarly for the grid and game objects.
- `match.state`. This method returns the state of the match as a namedtuple. The information of this method is to be used by the AI players. Example call: `state = match.state`. The state of the match contains the following accessible information.
  - `state.num_steps`: integer. The number of actions which have been performed.
  - `state.seat`: integer. The seat index of the current player. Before the game the players are randomly distributed over the seats and the seats are called in order. So the first turn the player in seat 0 is called, the second turn the player in seat 1 etc.
  - `state.player`: integer. The index of the current player
  - `state.winner`: integer. -1 if the match is still going or the index of the winning player if someone has won.
  - `state.area_players`: tuple. The players who own each area. `area_players[0]` gives the index of the player who owns the area with index 0.
  - `state.area_num_dice`: tuple. the number of dice on each area. `area_num_dice[0]` gives the number of dice on the area with index 0.
  - `state.player_areas`: tuple. the areas belonging to each player. `player_areas[0]` gives a tuple

with all the ares belonging to player 0.
  - `state.player_num_areas`: tuple. the number of areas belonging to each player. `player_num_areas[0]` gives the number of areas which player 0 owns.
  - `state.player_max_size`: tuple. The size of the largest cluster of areas belonging to each player. `player_max_size[0]` gives the largest number of adjacent areas of player 0.
  - `state.player_num_dice`: tuple. The total amount of dice of each player. `player_num_dice[0]` gives the total amount of dice of all the areas belonging to player 0.
  - `state.player_num_stock`: tuple. The number of dice which could not be placed on the areas of each player. The maximal number of dice on each area is 8. If a player should get more dice then he can place on an area, then these dice will go to his stockpile and he can place them on the next turn (or later) if possible. `player_num_stock[0]` gives the total number of dice which could not be supplied to player0's areas.
- `match.game.grid`: This method returns the grid of the match as a namedtuple to be used by the AI players. Example call: `grid = match.game.grid`. The grid of the match contains the following useful information.
  - `grid.areas[area_index].neighbors`. Gives the indices of all areas neighboring the area with index `area_index`.
- `match.step(action)`. Performs the action on the current state of the game and returns the new state. Example call: `grid, state = match.step(action)`.
  The Argument `action` is a tuple of two integers or `None`. The tuple of two integers should contain the following:
  - `from_area`: The index of the area from which the player wants to attack.
  - `to_area`: The index of the area to be attacked by the player.

  If `None` is passed as an action then the current player will end his turn.
  This function returns the new grid and state of the match (see match.game.grid and match.state respectively).
- `match.render()`. Renders the current state of the match.
  Some basic opponents are alo implemented to train against. I will list them all below.
- `dicewars.player.RandomPlayer()`: Plays randomly chooses one of the available attacks.
- `dicewars.player.AgressivePlayer()`: Always attacks when possible.
- `dicewars.player.WeakerPlayerAttacker()`: Attacks the neighbor only if the neighbor has less dice then him. It also sometimes (10% chance) attacks a player who is equally strong.
- `dicewars.player.PassivePlayer()`: Never attacks always ends his turn.
- `dicewars.player.DefaultPlayer()`: A somewhat smart player.

9 Install Tensorflow

There are several ways to install tensorflow/keras. Suggested method is using anaconda-navigator. Either in your current environment or in a new one (probably better)[1] open a Terminal and execute:

```
conda install -c conda-forge tensorflow
```

To test your install type in a jupyter notebook:

```
In [1]    import tensorflow as tf
          from tensorflow import keras

In [2]    tf.__version__
Out[2]    '2.3.0'

In [3]    keras.__version__
Out[3]    '2.4.0'
```

Your version numbers are likely higher.

[1] make sure you are on Python 3.10 or lower, but not 3.11