FUNDAMENTALS OF ARTIFICIAL INTELLIGENCE AND KNOWLEDGE REPRESENTATION

A.A. 2024/2025

Board Game Students Challenge 2024-2025

aka:

Tablut Challenge

Prof. **Michela Milano**

Dott. Andrea Galassi

with the support of Liam James, Gaetano Signorelli, Andrea Giovine, Federico Chesani, Allegra De Filippo

Objective of the competition

- To stimulate the comprehension and the discussion regarding the basic algorithms for solving games, in the context of AI discipline (for the teacher...)

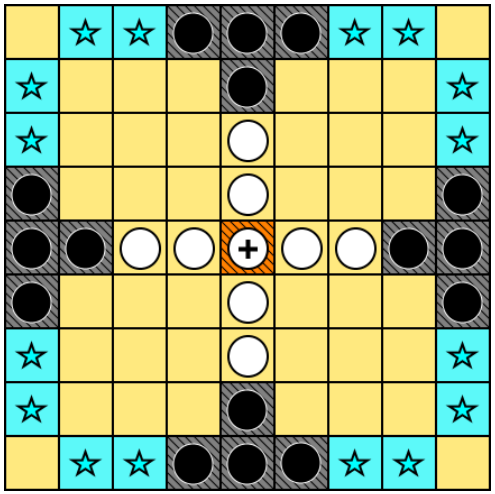
- TO WIN! (for the students…)

- **Acceptable solutions:** any solution that exploits algorithms somehow related to AI
  - Exploration of the state space
  - Genetic algorithms and swarm optimization
  - Neural Networks
  - Constraint-based approaches
  - Prolog-based solutions
  - …

Tablut – some info

- Northern Europe game, almost unknown
- There is no written trace of the original rules
  - Only written rules: documented by Linneo (the biologist) in Latin, after he had seen some Lapps playing it
- Tablut belongs to a family of medieval games known as Tafl Games
  - Other games: Hnefatafl, Tawlbwrdd, Brandubh
- **Asymmetric games**: players with different checkers and aims

Tablut – game overview



- Game board: grid of 9x9 squares
- Two players alternate in moving their checkers: attacker (Black) and defender (White)
  - White: 8 «Soldier» checkers and 1 «King» checker
  - Black: 16 «Soldier» checkers
- Checkers move orthogonally (like the Tower in chess)
  - Any amount of squares
  - Can’t pass on or over other checkers or obstacles
- A checker is "captured" (and removed from the game) if it is surrounded by opponent’s checkers on 2 opposite sides
- Aim of the white: make the King flee, reaching the side of the chessboard (any of the “escape tiles")
- Aim of the black: capture the King

Tablut – Game rules (Ashton rules) (1/5)

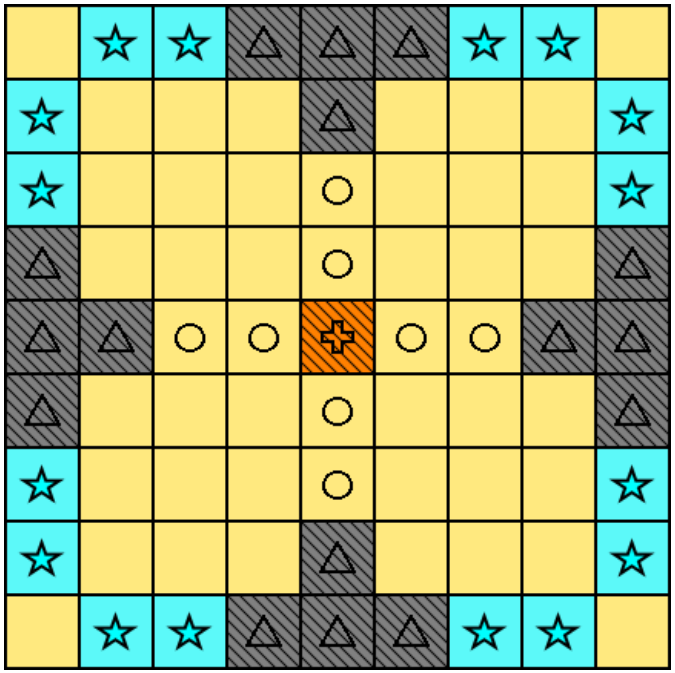

Castle


Camps

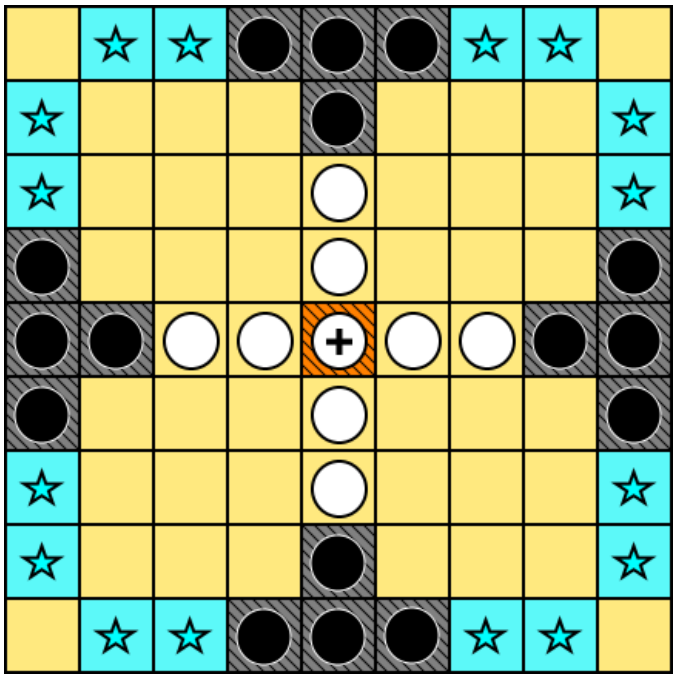

Escapes


Soldiers


King



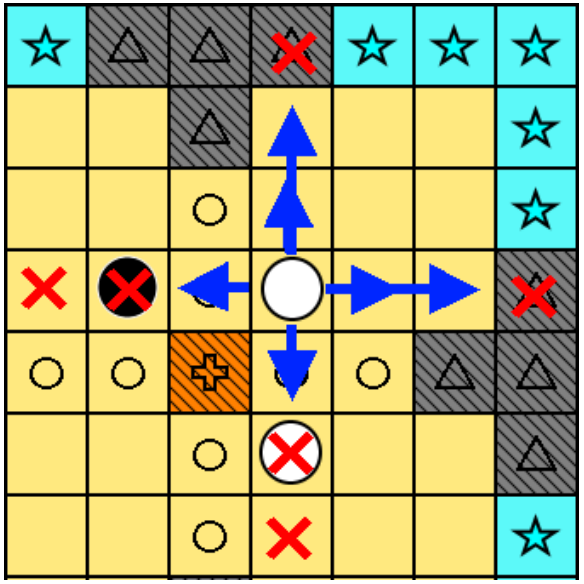
Game board



Initial position of the checkers

Tablut – Game rules (Ashton rules) (2/5)

- **Checkers movements**: orthogonal (up, down, left, right)
  - There is no limit in the number of cells that can be crossed
  - It is not possible to cross or end the movement on cells with Checkers, on the Castle, or on Camp cells
    - Exception! The black checkers can move in the cells of their starting Camp until they leave it. After that, they can't go back in.



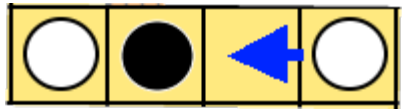
Blue arrows are legit moves for the white checker

Red X are illegal moves due to:

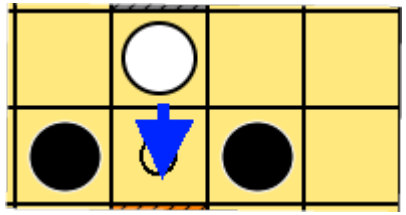
- presence of other checkers
- presence of camps

Tablut – Game rules (Ashton rules) (3/5)

- **Capture**: a Checker is captured if the opponent surrounds it with two checkers on opposite sides
  - It is possible to capture more Checkers at once
  - The capture must be "active”: if a Checker place itself in a surrounded position it is NOT considered captured



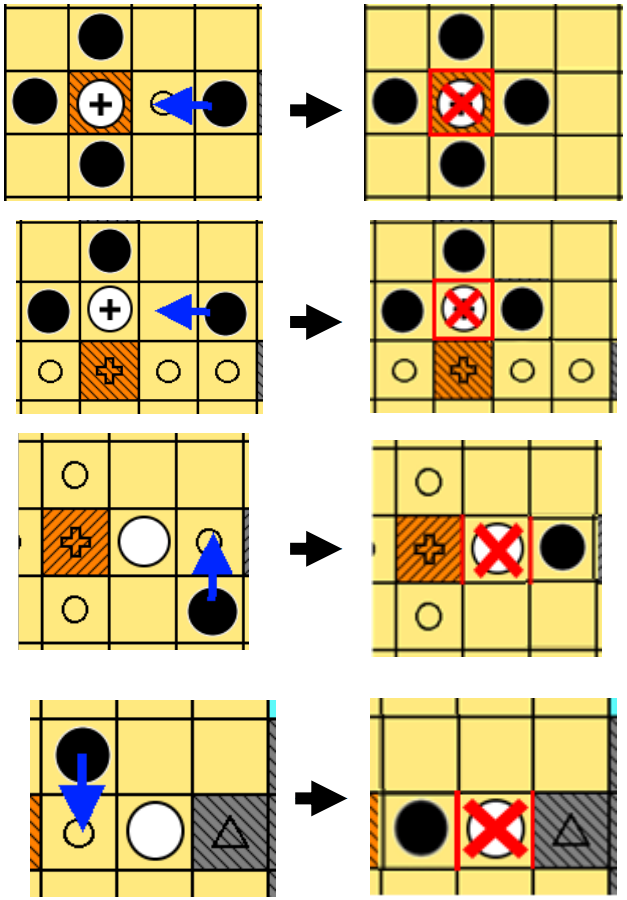
The white, by moving the checker, will capture the Black checker



The white checker is NOT captured, since there is no "active" capture

Tablut – Game rules (Ashton rules) (4/5)

- **Special cases of capture**
  - If the King is in the Castle, it must be surrounded on all the 4 sides
  - If the King is adjacent to the Castle, it must be surround on all the three free sides
  - If a Soldier is adjacent to the castle, it is sufficient to surround it with a checker on the opposite side of the Castle: the Castle acts as a «barrier». It doesn't matter if the King is in the Castle or not.
  - If a Checker (King or Soldier) is adjacent to a Camp, it is sufficient to surround it with a checker on the opposite side of the Camp: the Camps acts as a "barrier". It doesn't matter if the camp is occupied by a Checker or not



Tablut – Game rules (Ashton rules) (5/5)

- Start of the game: White moves first

- **End of the game**:
  - The King reaches an Escape tile: White wins
  - The King is captured: Black wins
  - A player can’t move any checker in any direction: that player loses
  - The same "state" of the game is reached twice: draw

Competition rules

- Separated tournaments: teams will be grouped accordingly to their dimension (i.e., the number of participants to the team).
- Round-trip matches
  - One game as White, one game as Black
- Every win, 3 points. Every draw, 1 point
- Limited time to choose a move:
  - 1 minute for each move
  - Timeout given by the referee (a server)
  - In case of timeout, the active player loses

What to make

- Students must create a software agent able to play a game, by communicating with the engine (the referee server, provided by the teachers)

- Communication messages are JSON strings

- A possible state representation is provided (useful, but its adoption is not mandatory)

Given tools (1/2)

Java Eclipse project on Github

https://github.com/AGalassi/TablutCompetition

Keep an eye on the Github page for bugs, bugfix, patches, etc.

The project contains:

- A server, that maintains the state of the game and communicates with players (State in *StateTablut.java*, server behaviour in *Server.java*)

- The game engine (*GameAshtonTablut.java*):
  - it checks if the moves (*Action.java*) are complaint with the rules
  - it updates the game state

- An abstract class (*TablutClient.java*), that offers two primitives:
  - send the selected move to the server
  - read the game state from the server

Given tools (2/2)

- A client that implements a textual interface to allow human players to play (*TablutHumanClient.java*)
- A client that implements a player that each turn makes a random move (*TablutRandomClient.java*)
- Clients that launch black and white players, by exploiting these classes
- A benchmark program to test rules and to realize personalized game states (*Tester.java*)
- Implementations of other versions of the game (probably broken!!!)

Communication between processes (1/2)

Communication is done through JSON strings

- At the beginning, the server waits for the players: the first player that must connect to the server is always the white player.
- Both the players communicate their names through strings.
  - Remember to write this part in your player
- Once both the players are connected, the server reacts by sending the current game state to both the players, then the game starts.
- Once the server has received the move from a player, it communicates the new state to both the players, than it waits for the move of the adversary.
- The player, after having sent its move to the server, must read from the server the new state (modified by its move). Then wait again for the new state (modified by its adversary move) through a blocking read function.

Communication between processes (2/2)

Summing up, a player "game cycle" is like to:

1. Read the game state (from the server);
2. Compute the move;
3. Send the move (to the server);
4. Read the new state (modified/updated by my move)
5. Back to 1. (read the state modified by the adversary – blocking read)

Adversary turn

- During the adversary's turn, a player can't execute computational activities

- Moreover, there is an upper bound to the amount of information stored in memory: hardware resources are limited.

Programming language suggestions

- **JAVA**: easy peasy, since all the project is in Java
- **Python**: Some libraries my require you to read 1 byte at time. There is a helper file in the main folder of the project that you can use to handle communication.
- **Other:** anything compatible with the machine on which the competition is running. The JSON strings should allow you to communicate between different implementations/languages
- Just in case: if you go for exotic choices, please drop a word to Galassi/De Filippo/Chesani/Giuliani/Maggio…
- We suggest you to use what you know. We will NOT help you with coding your player, you must be autonomous. In case you don't have much experience in programming, we highly suggest to group with someone who has more experience.

Techniques suggestions

- State space searches are the easiest thing to implement
  - Compact representations, symmetries, fast modification of the State, and fast heuristics are often keys to victory

- Do you want to study something new and fancy? Some hints could be genetic algorithms, neural networks, logic languages

Games dataset

- A dataset of previous game is available
  - http://ai.unibo.it/games/boardgamecompetition/tablut
- It might turn to be useful, to analyse and learn strategies
- It can be used for machine learning approaches

Requirements and constraints

- For each move the player has no more than 60 seconds. If it exceeds the deadline, it loses. This time may be lowered in case of necessity (for examples if there are too many participants to the competition)
- If the player proposes an "illegal" move, specific exceptions will be raised and the player will lose.

- **The agent must be launched specifying the following parameters (in the specified order):**
  - The role: «WHITE» or «BLACK».

    YES. The full word. Not B or W. Be smart and don’t be case sensitive.
  - The timeout time in seconds.
  - The server IP address.

- Your player can have additional optional parameters:
  - Optional! You should still define a default behaviour that the program will have during the competition

Discussion hints

- Try to realize your project following the principles of software engineering: re-usability, modularity etc.

- In case you would need to adapt your player to some variants of the game, what would you need to change? What could you re-use?

- Examples of variants:
  - Classic Tablut: no camps and there is another special capture
  - Brandubh: 7x7 grid, no camps, 4+1 whites, 8 blacks
  - Modern Tablut: like classic, but the escapes are in the corners and the king must be surrounded on each side

Presentations and questionnaires

- Once you have delivered your player, you will be asked to compile one anonymous questionnaires regarding your players and the techniques you have used.

- At the end of the competition, you will have to present your players to the professors and the other students. **The presentation is mandatory.**

- Towards the end of the competition, you will be asked to compile another anonymous questionnaires regarding your personal experience in the competition.

Organization

- The games will take place before the next winter break.
- Deadlines:
  - Players sent by the **November 30** (23:59, italian time) **on Virtuale**
  - Project presentation, discussion, and announcement of the winner: to be decided, but probably the last week of the course
  - Note: it is mandatory for all the team members to attend the final presentation
- Please subscribe the competition by the **13th of October**, using the module **on Virtuale**
- It will be possible to withdraw from the competition in any moment
- Some technical skills are required in order to implement the player –> please consider to have at least 1 member with a computer science background per teams

Virtual machine and player (1/2)

- We will give you a Virtual Machine where to put and develop your player.
  - https://liveunibo-my.sharepoint.com/:u:/g/personal/andrea_giovine_unibo_it/Eb_-2bR2YNtAs_F7D2i8jFkBY0KWWKfjNIY4-AoGMwVHFA
  - sha256sum: f6b3910448f87c09ff2063b663d789f7ab9cdeb0700ceb524cc6acd4e828a737

- You can access the Virtual Machine with credentials:
  - username: tablut password: tablut

- Do NOT modify these credentials! Nor any other configuration in the user

- Do NOT modify the SSH settings

Virtual machine and player (2/2)

- All the project resources should be included in a single folder named "/home/tablut/tablut"
- Your player should be launched by invoking a single script, named *runmyplayer.sh*
- In the virtual machine you will find an **example** of this
- To give you an idea, we will launch your player with a command similar to the following:
  - */home/tablut/tablut/runmyplayer.sh WHITE 60 192.168.20.101*
- PLEASE, test your player using a similar command

(OBVIOUSLY using the appropriate IP)

Deliver

- What to send and how:
  - Put the virtual machine image in your Onedrive account https://liveunibo-my.sharepoint.com/
  - Send through the VIRTUALE assignment a .txt file containing
    - The link for the virtual machine
    - How to execute your player from the command line
    - If your player makes use of optional parameters, please tell us and document these needs in this file
- We would like also to encourage you to use public and structured repositories for your project (such as Github, Bitbucket…).
  - This would make everything easier for us in case of problems, it would help you to organize your work, and learning to use them is quite useful

Hardware

Every player will run in an ad-hoc virtual machine (Virtual Box).

These details may change:

- The reference architecture is the latest Linux Debian 64 bit. Only this architecture will be considered, it will not be possible to use/exploit any other architecture
- Number of CPUs: 4
- RAM: 8GB
- Disk size: 30 GB
- No GPUs will be available (at run-time, it wouldn't make sense anyway).
- The player will be able to play using only local resources.
  - E.g., it will not be possible to query Google for the best move… no internet connection.
- PLEASE, BE SURE THAT YOUR PLAYER DOES NOT PRINT USELESS STUFF THAT SATURATES THE DISK

Troubleshooting,
or: things you should read carefully before submitting

Following these instructions will save up time for both you and us:

- While running the tournament, the VMs have no internet connection. Ensure that your VM contains all the dependencies you need to run your player, as you will not be allowed to download anything after submission
  - Generally speaking, using *pip/java* is enough to make things work, so try to avoid fancier stuff such as *pipenv* or *poetry* unless you don't really need it
- The script to run your player must be placed in the correct folder (*/home/tablut/tablut/runmyplayer.sh*). Please remove the random player that is already provided in the VM and put your player instead
  - Also, the name of your player will be used to refer to your group during the tournament, so please use a more self-explanatory name than "random"
- The script you provide must accept three parameters: (1) the role, (2) the timeout, and (3) the server IP. Please check that:
  - your code is case-insensitive (i.e., it works fine if we pass the string "wHitE" instead of "WHITE")
  - you can carefully handle timeouts, even if smaller than 60 seconds
  - you are not using a fixed IP (e.g., localhost) instead of the passed one

Doubts…

- All the rules can be changed in every moment, with a notification on the course website. Because Chesani is evil.
- Any feedback is appreciated ☺

- In case of doubts about the rules or the implementation, check the issues section in our Github, **or use the Tester class we provide**

- In case of need, please write to Chesani, Galassi, James, Signorelli and De Filippo.
  - **"AND"!!! not "OR". All of them!** And use the freaking "reply to all" in the emails to keep all the people in the conversation!!!

- In any case, Chesani has the final say
- Chesani IS ALWAYS RIGHT, and he HAS THE FINAL SAY