Index

1. Introduction

Welcome to the sixth edition of BEST LBG Zagreb's annual Artificial Intelligence Battleground hackathon. As you might have heard, this competition revolves around creating artificial intelligence bots that are used to play a competitive turn-based game for two players.

After the programming phase (which lasts for 20 full hours), the bots will face off against each other and the teams that built the 3 highest ranking bots will receive monetary prizes. However, you're not doing this for the prize alone, as tech company representatives will observe and evaluate your work during the hackathon. Play your proverbial cards right and you might just land a job...

Please read this document in its entirety and pay special attention to section 2, General concepts, as it contains explanations of (hopefully) everything relevant about the game's internal logic. Of course, if you have any questions, you can ask the organizers and we'll try our BEST (pun pun) to answer them.

Good luck!

2. General concepts

The concepts described here serve to provide you with a clear understanding of how the game itself works. Please read this section carefully, as we've tried to cover all relevant information. For any further questions, just grab your nearest BEST member with a "topic team" ID card, or, if you feel your question is too specific, the guy with the "topic responsible" card.

- At the **start of each match**, each bot **chooses 12 creatures** for their "creature pool". The pool represents all of the creatures available to the bot during the course of the match. This phase is referred to as "*POOL_SELECTION*" in the code.
- A match consists of up to **5 rounds**.
- Each round begins with a **pre-round selection phase** called "*ROUND_SELECTION*" during which the bots pick creatures to use in that round. A maximum of **4 creatures** can be used per round.
- The creatures will fill slots in the order they are selected - creatures selected first will be at the front of the line. Only the **first creature (slot 0)** can attack at any given moment.
- After both bots have picked their lineups for the next round, the round begins (the game transitions to "*ROUND_IN_PROGRESS*").
- Rounds consist of **turns**. Each turn, the bots must send their chosen move to the server.
- The following moves are available: **attack** - attacks using creature at the front (slot 0); **buy item** - buys and instantly uses chosen item if the bot has enough atoms; and **swap** - swaps the creature in the front with the creature in the specified slot (if that creature is alive). More on moves in section 7.
- Move execution order depends on **move priority first** and **front creature speed second**. Swap actions are executed before any others, then item use, and finally, attacks. If both players' moves are of the same priority, the one whose front creature has higher speed is

executed first. If both move priorities and speed stats are equal, the moves are done simultaneously.

- Creatures have 5 stats: **element** (water, fire, nature), **speed**, **health**, **damage** and **attack type** (melee, range, area of effect).
- As in almost every game of this kind: **water beats fire, fire beats nature, nature beats water**; attacking with a “stronger” element deals 1.2x damage, while having a “weaker” element multiplies the damage by 0.8x.
- Creatures are balanced around their attack type: **melee** creatures have the highest damage stat, but can only target their enemy’s front creature (slot 0); **ranged** creatures do the least damage, but can target enemy creatures on any slot; **area-of-effect** creatures deal damage to all enemy creatures when they attack, but the damage value is reduced the further away the enemy creature stands (slot 0: 80% damage, slot 1: 60% damage, slot 2: 35% damage, and slot 3: 15% damage).
- If a **creature dies in a round**, it is moved to the last slot and can no longer be used for the duration of the match. However, a **creature that survived the round** has its health replenished and can be used again in the next round.
- Rounds are declared as **draws** if both players lose all of their slotted creatures or the turn limit expires (to prevent endless rounds). The turn limit is set to 600 turns.
- The **match winner** is the bot who **has won more rounds**. The match ends **after 5 rounds** or when **one bot has no more living creatures**.
- At the start of each round, three items are generated in the “**shop display**” and the bots are given a certain amount of atoms (currency) with which they can buy damage, health and speed upgrades, damage/healing items or domains from the display.
- The same item is never generated twice and bots can buy any given item **once**. However, **both bots are able to buy the same item** (shop displays are identical for both players).
- Atoms provided per round are as follows: 3, 5, 7, 9 and 11 (for rounds 1

through 5). Unspent atoms **do not carry over** to between rounds.

- Upgrades can be applied to any living creature in the current round. Upgrade items can provide flat value or percentage bonuses and also differ in scope: **single target** items must be provided with a target slot argument upon use, while **multi-target** items affect all specified creatures. Along with single and multi-target, items can be defined as "friendly" (affects own creature/s), "enemy" (affects enemy creature/s) or "all" (ex. an "all creatures multi-target" item's effect extends to both friendly and enemy creatures).
- Damage items deal **flat elemental damage** according to their specified element and value to creatures within their scope. Healing items heal in the same way. However, **elemental weaknesses are reversed** for healing, meaning water is considered "strong" when healing a nature creature (1.2x multiplier), nature is strong when healing fire etc. Healing a creature of the same element, as with elemental damage, multiplies the amount by 1.0x.
- Domains change the battlefield until replaced with another domain (or the round ends). Domains have a specified element and **affect all creatures** (both friendly and enemy) of the **same element**. If a domain has its element set to "*NONE*", it affects all creatures of all elements.
- Domains come in two main varieties: **elemental mod** domains multiply the damage done by the specified element by a multiplier defined in the domain object, while **turn action** domains heal or deal damage to all creatures of the same element (all if the element is *NONE*) each turn.
- The bot determines what to do by looking at the opponent's remaining creatures (out of a maximum of 4), their order and their stats, as well as that round's available shop items.

3. Main objectives

The main objective of the competition is to create a bot capable of competitively playing the game described in the previous section. This bot

must perform the required actions as if it were a human agent. After the development phase of the hackathon is over, we will pit the bots against each other. The team that created the winning bot will receive the main prize, followed by the teams whose bots placed 2nd and 3rd.

4. Implementation and requirements

The project consists of three main components: server, game logic and display. The server handles user authentication as well as any user commands, which are passed on to the game logic component for processing, and generates and sends responses depending on which command was received. All interaction with the game logic component is done through the server API. Both the server and the game logic were written in the Java programming language and running them locally requires installation of Java 17.

The display component periodically pulls game state information from the server and generates visual representation. The component was created in Unity.

5. User registration

Before sending commands to the server, you will have to register with the authentication component. This is mandatory in order to avoid situations in which a third party poses as a player in an ongoing game and sends malicious commands.

Registration is done by sending a POST request to the server at **/public/register**. The request must contain username and password parameters. The server will respond with your unique bearer authentication token which you will use when connecting to the web socket.

The token has a lifetime of 24 hours. However, you can request a new token at any time by sending a POST request containing the specified username and password parameters to the server at **/public/login**.

6. Connecting to the web socket

Connect to **ws://<server-domain>:<port>/player**. The connection request must contain an authorization header with your bearer token in the format of “Bearer <token>“.

We have prepared an example project (written in Java) to serve as a tutorial of sorts for connecting to the web socket. The project zip can be downloaded here. You can use any programming language you want.

7. Sending commands and receiving responses

User commands must be sent to **/websocket/game** in JSON format. The JSON must have the following form:

{
gameId : Long
messageType : String
messageText : String
}

where gameId is the identifier of the game the user is a player in, messageType represents the type of the command (get game phase, get number of victories, make move etc.), and messageText contains arguments required for specific commands.

To receive responses from the server, users must subscribe to **/user/queue/player**. The server responds with JSON messages with the following form: *{ gamePhase : String, serverMessage : JSON }*. The contents of serverMessage vary depending on the type of user command sent.

MessageType **must be** one of the following values:

- GET_POSSIBLE_CREATURES - gets all creature types,
- SELECT_PLAYER_CREATURE_POOL,
- GET_PLAYER_CREATURE_POOL - gets all creatures in pool,
- SELECT_PLAYER_CREATURES_IN_PLAY,
- GET_PLAYER_CREATURES_IN_PLAY - gets all of the player's slotted creatures (during round),
- GET_OPPONENT_CREATURES_IN_PLAY - gets all of the opponent's slotted creatures (during round),
- GET_SHOP - gets all shop items in the display,
- GET_AVAILABLE_SHOP - gets all shop items in the display the player hasn't bought yet,
- GET_CURRENCY - gets available atoms,
- GET_DOMAIN - gets currently set domain (null if none),
- GET_ROUND_NUMBER,
- GET_ROUNDS_WON,
- GET_GAME_PHASE - gets current game phase, and
- SELECT_MOVE.

When selecting a **creature pool**, messageText must contain exactly 12 creature type names separated by spaces. Types can be repeated.

When selecting a **round lineup**, messageText must contain between 1 and 4 creature identifiers (*long* value). IDs must be unique.

When selecting a **move**, messageText must contain a move designation ("attack", "item" or "swap") depending on the action the player wishes to make and any additional arguments specific to the move action, separated by spaces.

Attack moves can be followed by a single slot designation (between 0 and 3 if all 4 enemy slots are full). If the attacker is a ranged creature, it will attack the creature in the specified slot. If left unspecified, it will default to slot 0. If there is a specified target slot and the attacker is not ranged, it will simply be ignored.
> Ex. "attack 1" attacks slot 1 if the attacking creature is ranged, otherwise it attacks slot 0.

Item moves must be followed by the name of the item the player wishes to buy. If the item has a single target effect, the player can also specify a target slot (friendly or enemy, depending on item scope). If the

slot is left unspecified, it defaults to slot 0. If the item's effect is multi-target, the slot specification is ignored if provided.
> Ex. "item smallrock 2" uses "SmallRock" on slot 2 (in this case, the enemy's slot). Note that the item must exist in the shop display and must not have already been bought by the player in question.

Swap moves must be followed by a target slot index. Upon executing a swap move, the front creature (slot 0) is swapped with the creature in the specified slot, if the specified creature is alive.
> Ex. "swap 2" swaps the player's front creature and their creature at slot 2.

8. Final words

We wish you luck! Remember: the first rule of AIBG is to have fun and be yourself!

- the AIBG 6.0 team, BEST Zagreb