Bot!Battle! Game Display Documentation

Each game challenge will have to produce two JSON messages that will be described in further detail

1. Game Initialization Message
2. Turns Data

Other things that must be done when creating a game

- Place images on the website, ideally in a folder within the ‘games’ folder to hold all images/resources
- Create any template bots for the challenge
- Create a default bot and upload it into the database

Game Initialization Message

```
{
        background: <image_name>,
        defaultTimestep: <timestep>,
        defaultBot: <bot_id>,
        imagesToLoad: [<image_1>, <image_2>, ...],
        entity: [<entity_1>, <entity_2>, ...]
}
```

where….

<image_name> is the filepath of the background image to load, ex. "assets/images/background.png". The game window is 800x600, and the image will be scaled to fit.

<timestep> is a floating point number representing the default length of a turn in seconds. All of the animations of moves will be done relative to this timestep. As mentioned below, there is the option to make a turn longer or shorter. In addition, there will be controls on the web page that allow the user to speed up or slow down the playback of the match. This speed up or slow down will be done relative to this default turn length.

<bot_id> is an integer number of a default bot for players to use for this challenge corresponding to an id in the user_bots table.

<image_#> is an image that needs to be loaded into the game. Each of these images will use the following format.

```
        {
                imagePath: <path>,
                name: <image_name>
        }
```

<path> is the url/path that the file is located at, ex. "games/checkers/basic_red.png"

<name> is a string representing the name of the image, ex. “basicred”

<entity_i> defines the ith game entity. These game entities should be added to the game in the order specified in this array, so that later entities will appear on top of earlier entities, should they overlap.

Each of the <entity_i> records will have the following format:

```
        {
          id: <id>,
          type: <type>
          visible: true|false,
          initX: <x_coord>,
          initY: <y_coord>,
          width: <width>,
          height: <height>,
          flipped: true|false,
          rotation: <degrees>,
          value: <value>
        }
```

<id> is an integer that uniquely identifies the entity.

<type> is either "text", "object", "spriteBunny", "spriteChicken", "spriteZombie", or "spriteAlien”.

visible: A boolean value which specifies the objects initial visibility.

<initX>: A float value which specifies the initial top left x coordinate of the object.

<initY>: A float value which specifies the initial top left y coordinate of the object.

<width>: A float value which specifies the initial width for the object.

<height>: A float value which specifies the initial height for the object.

flipped: A Boolean value which indicates if the object is flipped along the vertical axis.

<degrees>: A float value which indicates the rotation of the object.

If type = “text”, then the following additional type-specific arguments are supported:

value: String representing the text to be displayed.

font: String representing the style and size of the font, ex. ‘bold 20pt Arial’.

backgroundColor: The color of the background of the text object.

fill: Hex value indicating the color of the text, ex. "#00FF00".

If type = object, then the following type-specific arguments should be supported:

value: Name of the image, as specified in imagesToLoad.

Turns Data

```
[
        {
                timescale: <timescale>,
                currentPlayer: <curr_player>,
                nextPlayer: <next_player>,
                stdin: <std_in>,
                stdout: <std_out>,
                stderr: <std_err>,
                turnChanges: [<config_1>, <config_2>,…],
                <GAME STATE INFO>
        },
        {….},
        {….},
        …
]
```

where…

<timescale> is a scale to be used for the current turn

<config_i> is a change specification

<curr_player> is an integer corresponding to the bot that has made its move during this turn

<next_player> is an integer corresponding to the bot that will make the next move

<std_in> <std_out> <std_err> are messages that will be displayed to the user. These can be strings or arrays of strings, or can be omitted entirely.

stdin: this should be what was given to the bot this turn
stdout: this should be what the bot produced as output
stderr: this should hold anything that the bot wrote to standard error

<GAME STATE INFO> is info that the display will ignore, if needed by the game evaluation team, it can be added here.

Each change specification (<config_i>) will have the following format

```
{
        id: <id>,
        changes: [
                {
                        <arg_1>: <arg_1 value>
                        <arg_2>: <arg_2 value>
                        (etc…)
                },
                ...]
}
```

where

<id> Is the id of one of the entities defined in the Game Initialization Message

changes Can include any of the following:

> **start**: A number between 0 and 1 that indicates when relative to this timestep to initiate the change, e.g. if it is 0, the action should be initiated at the start of the timestep, if it is 0.5 it should be initiated halfway through. **This is the only argument which cannot be omitted.** If the entity has multiple changes in one turn, each successive change must have a start time equal to or greater than the end time of the previous change (or the start time of the previous change if it has no specified end time).
>
> **end**: A number between 0 and 1 that indicates when to finish the change. This is only applicable for animated changes like moves, for changes such as changing the value of text you only need to specify the start.
>
> **action**: Specifies an action for the entity. If the entity is a sprite, this can be "walk", "fall", "attack", or "defend" to determine animation. It can also be a special action such as "jump".
>
> **x**: Specifies an x position for the entity to move to in pixels, with 0 being the leftmost part of the screen.
>
> **y**: Specifies an y position for the entity to move to, with 0 being the topmost part of the screen.
>
> **width:** Specifies the width of the entity at the end of the change.
>
> **height:** Specifies the width of the entity at the end of the change.

rotation: Specifies how far the entity should rotate during the change.

flipped: A boolean value which specifies whether the entity sprite is flipped.

visible: A boolean value which specifies whether the entity is able to be seen.

initX, initY, initWidth, initHeight, initRotation: Arguments which specify initial values for the change. For instance, setting x=500 will make the entity move to x=500 at the end of its turn, whereas setting initX=500 will instantly place the entity at x=500. It is not recommended to include these settings in the change specification.

For the "text" type:

value: String representing the text to be displayed.

font: String representing the style and size of the font, ex. ‘bold 20pt Arial’.

backgroundColor: The color of the background of the text object.

fill: Hex value indicating the color of the text, ex. "#00FF00".

For the "object" type:

value: Name of the image, as specified in imagesToLoad.

Additionally, a change can be used to dynamically create a new object using the **create** argument. The create argument should contain the initialization message for the entity, and the change should include only start and create. For instance:

```
{
	"id": 105,
	"changes": [{
		"start": 0.3,
		"create": {
			"id": 105,
			"type": "spriteMummy",
			"visible": true,
			"initX": 500,
			"initY": 400,
			"width": 50,
			“height": 50,
			“flipped": true,
			"rotation": 0
		}
	}]
}
```