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

THIS IS A LIVING DOCUMENT! IT IS STILL SUBJECT TO CHANGE! I WILL POST ANY CHANGES AS SOON AS THEY ARE MADE.

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 name given to the background image for the game (see imagesToLoad)

<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 animation. 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 the user_bots table.

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

{
imagePath: <path>,
name: <image_name>,
imageType: <image_type>,
height: <image_height>,
width: <image_width>
}

<path> is the url/path that the file is located

<name> is the name the image will be refered as – see <image_name> in Game Initialization Message

<imageType> is a string that is either ‘spritesheet’ or ‘image’

<height> and <width> are integer values cooresponding to the height and width of the image. THESE ARE ONLY REQUIRED IF <imageType> IS ‘spritesheet’!!!

<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>,
  <args>
}
```

<id> is an integer that uniquely identifies the entity

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

visible specifies the objects initial visibility

<initX> is the initial top left x coordinate of the object

<initY> is the initial top left y coordinate of the object

<width> is the initial width for the object

<height> is the initial height for the object

flipped indicates if the object is flipped along the vertical axis

<degrees> indicates the rotation of the object

<args> are type-specific arguments

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

value: text to be displayed
font: the style and size of the font – example ‘bold 20pt Arial’
fontStyle: the style of the font – example ‘italic’ or ‘bold’
fontWeight: the weight of the font – example ‘bold’
fontSize: the size of the font – example ‘32px’
backgroundColor: the color of the background of the text object
fill: the color of the text

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

value: url of the image

Turns Data

```
[
        {
                timescale: <timescale>,
                turnChanges: [<config_1>, <config_2>,…],
                currentPlayer: <curr_player>,
                nextPlayer: <next_player>,
                stdin: <std_in>,
                stdout: <std_out>,
                stderr: <std_err>,
                <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 it's 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 all strings of messages that will be displayed to the user

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: [
                {
                        action: <verb>,
                        start: <starttime>,
                        end: <endtime>,
                        <args>
                },
                ...]
}
```

where

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

<verb> is a type-specific verb

<starttime> is 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

<endtime> is a number between 0 and 1 that indicates when to finish the change
(this is only applicable for animated changes like moves)

<args> are verb-specific arguments

All types support the following verbs:

show

hide

move (with arguments): x*, y*, width, height, rotation, flipped
        *denotes required argument

For the "text" type:
(For some examples, see text of Game Initialization Message)

setText: change the text displayed to the value field
setFont: the style and size of the font
setFontStyle: the style of the font
setFontWeight: the weight of the font
setFontSize: the size of the font
setBackgroundColor: the color of the background of the text object
setFill: the color of the text

For the "sprite..." types:
Each of these have the following argument options...
x, y*, width, height, rotation, flipped*
**denotes required argument*

walk: animate walking for the sprite

fall: animate falling for the sprite

attack: animate an attack

defend: animate a defense

For the "object" type:

setImage: change the image of the object