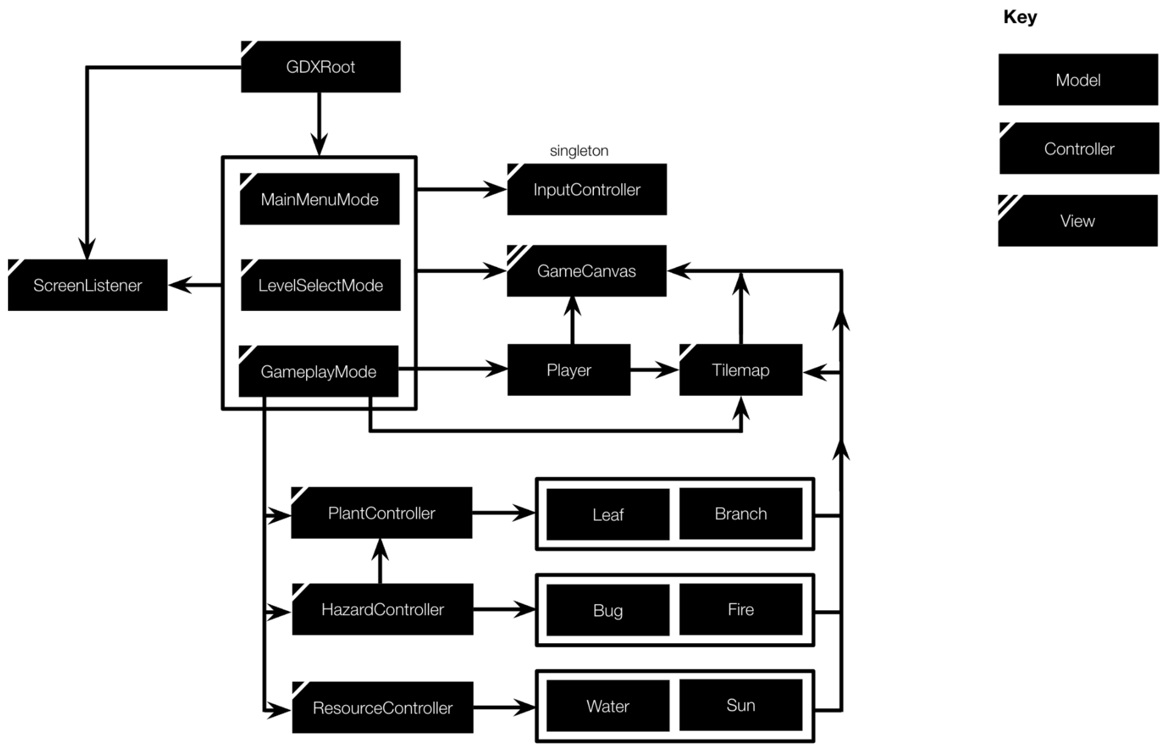
# *PHYTOPOLIS* ⍰ Architecture Specification

**Syndic8** — Team 10

Ireanne Cao, Kevin Chang, Alanna Cooney, Shirley Li, Amy Mai,
Tawakalt Okunola, Pedro Pontes García, Jordan Rudolph

## Dependency diagram



## Models

### Branch

**Description:** This class represents a structural branch that the player has built. It stores information about its location.

**Justification:** This model provides an easy way to represent branches, which are a core part of our game.

| **Responsibilities** | **Collaborators** |
|---|---|
| Get level scaling parameters | Tilemap |
| Draw Branch | GameCanvas |

### Leaf

**Description:** This class represents a leaf platform that the player has built and can stand on. It stores information about its location. It can be upgraded to a bouncy leaf and it can be eaten by a bug.

**Justification:** This controller provides an easy way to represent leaves, which are a core part of our game.

| **Responsibilities** | **Collaborators** |
|---|---|
| Get level scaling parameters | Tilemap |
| Upgrade to bouncy | - |
| Get/set eaten status | - |
| Draw Leaf | GameCanvas |

## Bug

**Description:** This class represents a cyberbug that eats leaves. It stores information about its location and its remaining lifespan.

**Justification:** There may be multiple bugs on the level at a time, and therefore it is reasonable that their location be handled by a model class. In addition, they have a lifespan after which they disappear, so their remaining lifespan needs to be stored separately for each bug present on the level.

| **Responsibilities** | **Collaborators** |
|---|---|
| Get level scaling parameters | Tilemap |
| Get/set location | |
| Get remaining lifespan | - |
| Update timers | - |
| Draw Indicator | GameCanvas |

## Fire

**Description:** This class represents a fire that threatens to burn the plant if not extinguished. It stores information about its burning status, duration, and location.

**Justification:** There can be multiple fires with different locations. Additionally, this allows us to adjust how fast the fire takes to burn, and track if it has been extinguished.

| **Responsibilities** | **Collaborators** |
|---|---|
| Get level scaling parameters | Tilemap |
| Get/set location | - |
| Get/set duration | - |

| **Responsibilities** | **Collaborators** |
|---|---|
| Get/set burning status | - |
| Update timers | - |
| Draw Fire | GameCanvas |

## Water

**Description:** This class represents a resource collection point, which may grant water to the player when collided with. It stores information about its current supply, maximum supply, location, and cooldown.

**Justification:** There can be water at multiple different locations, each of which could provide a different amount of water. The cooldown for being able to pick up this resource incentivizes the player to leave and come back rather than just staying and maxing out their resources.

| **Responsibilities** | **Collaborators** |
|---|---|
| Get level scaling parameters | Tilemap |
| Get/set location | - |
| Get current supply | - |
| Get maximum supply | - |
| Update timers | - |
| Draw Water | GameCanvas |

## Sun

**Description:** This class represents a resource that adds extra time to the level timer. It stores information about its physics properties as well as its transparency in order to perform the fadeout as it disappears.

**Justification:** This class allows the game to keep track of multiple suns falling from the sky at different locations. Additionally, it allows for the fadeout behavior to be self-managed by each sun.

| **Responsibilities** | **Collaborators** |
|---|---|
| Get level scaling parameters | Tilemap |
| Start fadeout based on plant parameters | - |
| Update physics properties | - |
| Update timers | - |
| Draw Sun | GameCanvas |

## Player

**Description:** This class represents the player in the game. It stores information about the player's position, movement, plant growing and hazard management (whether a fire can be put out) abilities.

**Justification:** The player is different from other models as it performs physics tasks such as jumping and dropping. We also need a way to track whether a player can resolve hazards or grow plants as other models should not be able to perform these tasks.

| **Responsibilities** | **Collaborators** |
|---|---|
| Get level scaling parameters | Tilemap |
| Get position | - |

| **Responsibilities** | **Collaborators** |
|---|---|
| Get velocity | - |
| Apply force | - |
| Apply physics to the player | - |
| Draw Player | GameCanvas |

## Controllers

### GDXRoot

**Description:** This is the root controller. It creates the mode controllers, updates them, and draws them to the canvas. It controls mode transitions.

**Justification:** This controller is the base for all other controllers, so it is essential to the game.

| Responsibilities | Collaborators |
|---|---|
| Initialize canvas | GameCanvas |
| Initialize modes | MainMenuMode, LevelSelectMode, GameplayMode |
| Update modes | MainMenuMode, LevelSelectMode, GameplayMode |
| Draw modes | MainMenuMode, LevelSelectMode, GameplayMode, GameCanvas |
| Transition between modes | MainMenuMode, LevelSelectMode, GameplayMode, ScreenListener |

### ScreenListener

**Description:** This class is responsible for bringing the modes and GDXRoot together so that GDXRoot knows when to switch screens/modes.

**Justification:** This class is needed for GDXRoot to know when to switch screens and which screen to switch to. It provides a way for the modes to communicate with GDXRoot about when the mode should be exited.

| Responsibilities | Collaborators |
|---|---|
| Set the current mode | - |

## MainMenuMode

**Description:** This class is responsible for displaying the screen that users will see when loading our game. The player can choose to play the game which takes them to the LevelSelectMode or click on the Setting Button to display information about the gameplay.

**Justification:** The player needs an introduction to the game, where they can choose to learn how to play the game or go straight into selecting a level.

| **Responsibilities** | **Collaborators** |
|---|---|
| Load assets | - |
| Display title screen | GameCanvas |
| Play music | - |
| Open menus based on player input | InputController |
| Change graphics and input settings | GameCanvas, InputController |
| Notify listener to change mode when user makes selection | InputController, ScreenListener |

## LevelSelectMode

**Description:** This class is responsible for displaying the screen to select an unlocked level in the game.

**Justification:** The player needs a way to select a level, since they will not necessarily start at the last level played, and may want to revisit an unlocked level.

| **Responsibilities** | **Collaborators** |
|---|---|
| Display level selection | GameCanvas |
| Play music | - |

| **Responsibilities** | **Collaborators** |
|---|---|
| Switch between level screens based on player input | InputController |
| Notify listener of change in scene when user makes selection | InputController, ScreenListener |

## GameplayMode

**Description:** This class is responsible for handling all the gameplay during the actual game. This includes performing all the activities on the activity diagram in order, owning subcontrollers and models to perform those activities.

**Justification:** This class is the root of the main game. It  initializes the controllers that allow for a level to be played.

| **Responsibilities** | **Collaborators** |
|---|---|
| Initialize level | PlantController, ResourceController, HazardController, Player, Tilemap |
| Notify Player of a change in movement | Player, InputController |
| Update resource amounts | ResourceController |
| Add/remove hazards | HazardController, InputController |
| Notify PlantController to grow new branches and leaves | PlantController, InputController |
| Display game screen and game objects | GameCanvas |
| Notify listener of change in scene when exiting mode | InputController, ScreenListener |

## PlantController

**Description:** This class is responsible for handling the state of the plant, including all branches and platforms the player has grown.

**Justification:** This controller provides a clean and simple way to make changes to and access information about the plant.

| **Responsibilities** | **Collaborators** |
|---|---|
| Create/destroy branches and leaves | Leaf, Branch |
| Initialize plant structure | - |
| Draw Leaf and Branch objects | Leaf, Branch |

## HazardController

**Description:** This class is responsible for hazard generation and updates, which include controlling the fire spreading and drone trajectory.

**Justification:** This controller is important for handling the state of hazards in the game, including spawn rate, spawn time, and whether hazards are to be destroyed.

| **Responsibilities** | **Collaborators** |
|---|---|
| Initialize hazards | Bug, Fire |
| Update hazard duration/phase/location | - |
| Spread/stop spreading fire | Fire |
| Destroy branches and leaves | PlantController |

## ResourceController

**Description:** This class is responsible for hazard generation and updates, which include controlling the fire spreading and drone trajectory.

**Justification:** This controller is important for Initializing Sun and Water resources and updating their counts. It is essential for game balance and difficulty ramping.

| **Responsibilities** | **Collaborators** |
|---|---|
| Initialize resources | Water, Sun |
| Update resource supply/location | Water, Sun |
| Increase/decrease counts of resources | - |

## InputController

**Description:** This class is responsible for hazard generation and updates, which include controlling the fire spreading and drone trajectory.

**Justification:** This controller centralizes user input handling and translates input into flags understandable by the game modes. Since the controller is a singleton used by other classes, it synchronizes the input during a frame for all classes that make changes based on input.

| **Responsibilities** | **Collaborators** |
|---|---|
| Synchronize user input | - |
| Get/set whether to grow a branch at a certain direction | - |
| Get/set whether to display the game settings | |
| Get/set whether the player can switch to a different screen | - |
| Get/set Player movement | - |

| Responsibilities | Collaborators |
|---|---|
| Get/set whether the player wants a leaf to be added | - |
| Get/set whether the player wants to put out a fire | - |

## Tilemap

**Description:** This class is responsible for populating levels from tilemap files, drawing the tiles associated with those levels, and providing getters for tilemap parameters.

**Justification:** This controller is essential to load our level files, described in a separate section.

| Responsibilities | Collaborators |
|---|---|
| Get level parameters | - |
| Populate level | - |
| Draw level | GameCanvas |

## View

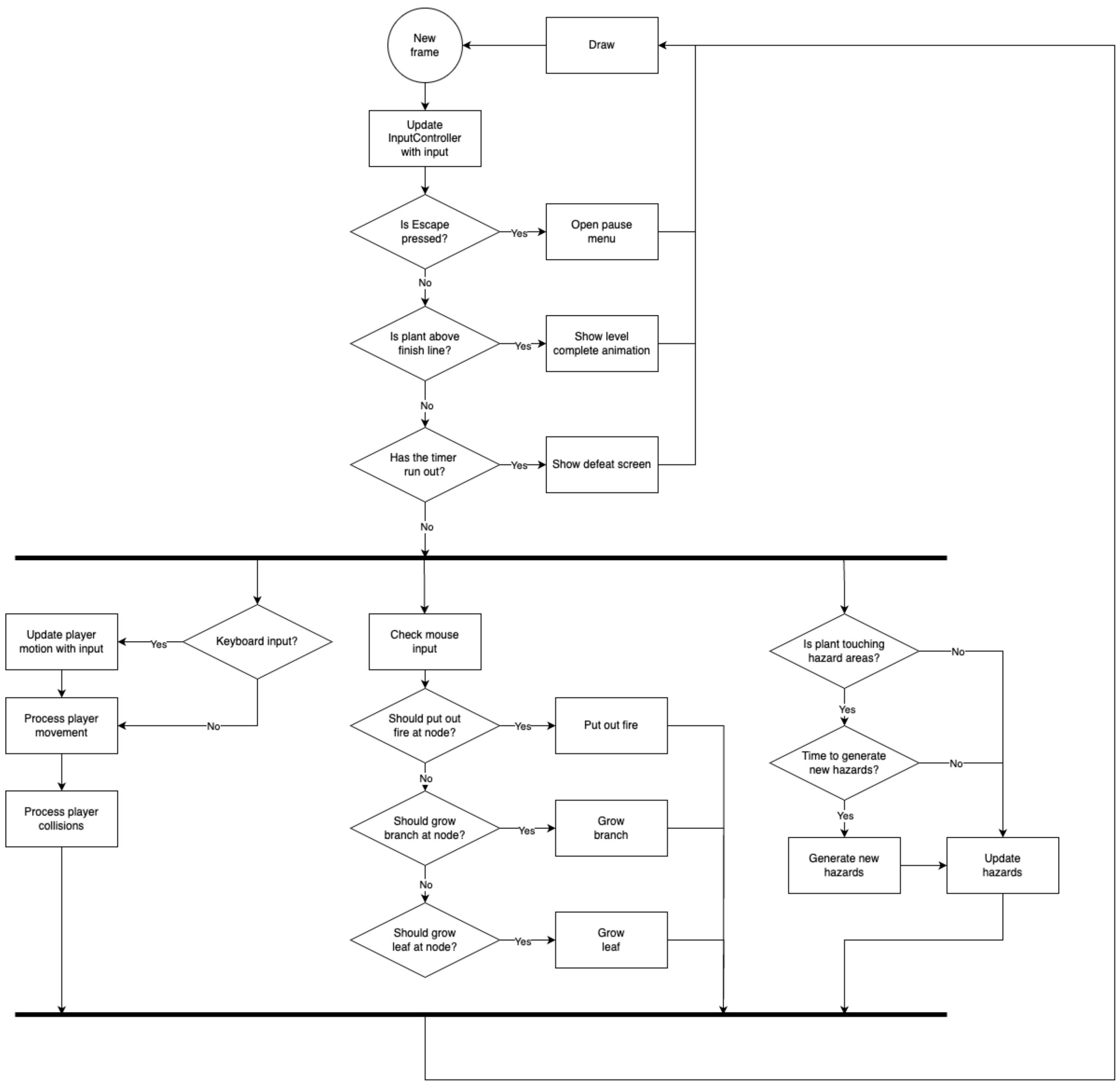
### GameCanvas

**Description:** This class is responsible for drawing to the screen. It handles all the drawing of our game, and includes a master draw method that is called by the models.

**Justification:** This is one of the core classes of our game and is our only view class. All models that have draw methods will use the GameCanvas master draw method.

| **Responsibilities** | **Collaborators** |
|---|---|
| Initialize and close SpriteBatch | - |
| Draw sprites to the screen | - |

## Activity diagram


New frame
Draw
Update InputController with input
Is Escape pressed?
Yes
Open pause menu
No
Is plant above finish line?
Yes
Show level complete animation
No
Has the timer run out?
Yes
Show defeat screen
No
Update player motion with input
Yes
Keyboard input?
Process player movement
No
Process player collisions
Check mouse input
Should put out fire at node?
Yes
Put out fire
No
Should grow branch at node?
Yes
Grow branch
No
Should grow leaf at node?
Yes
Grow leaf
Is plant touching hazard areas?
No
Yes
Time to generate new hazards?
No
Yes
Generate new hazards
Update hazards

## Data representation model

### Save file

The save file will be saved in JSON format at the directory specified for the operating system for application data storage[1], inside a subdirectory named “Phytopolis”. It will contain information about the last beaten level, and the best completion times recorded for each level. The JSON file contains the following keys:

- “lastBeaten” (integer): the zero-based index of the last beaten level.
- “bestTimeX” (float): the best completion time in seconds for level “X”, -1 if not beaten.

**Example:**

```
{
  "lastBeaten": 6,
  "bestTime1": 8.081565856933594,
  "bestTime2": 11.679478645324707,
  "bestTime3": 15.932108879089355,
  "bestTime4": 16.054201126098633,
  "bestTime5": 100.47856140136719,
  "bestTime6": 69.61925506591797,
  "bestTime7": 77.17472839355469,
  "bestTime8": -1,
  "bestTime9": -1,
  "bestTime10": -1,
  "bestTime11": -1,
  "bestTime12": -1
}
```

### Settings file

The settings file will be saved in JSON format at the directory specified for the operating system for application settings storage[2], inside a subdirectory named “Phytopolis”. It will contain information about the controls and the graphics settings. The JSON file contains the following keys:

[1] On Windows, the path is represented by the environment variable “APPDATA”. On macOS, it is the path “Library/Application Support/” under the local user’s home directory. On Linux, it is represented by the environment variable “XDG_DATA_HOME”, and defaults to the path “.local/share/” under the local user’s home directory if the variable is undefined.

[2] On Windows, the path is exactly the same as the save file. On macOS, it is the path “Library/Preferences/” under the local user’s home directory. On Linux, it is represented by the environment variable “XDG_CONFIG_HOME”, and defaults to the path “.config/” under the local user’s home directory if the variable is undefined.

- “xKey”/”xButton” (integer): the LibGDX key code mapped to action “x”, -1 if not set.
- “resolutionIndex” (integer): the index of the graphics display mode within the array of graphics display modes provided by the main display.
- “fpsIndex” (integer): the index of the frame rate within the array of valid frame rates (VSync, 15fps, 30fps, 45fps, 60fps, 90fps, 120fps).
- “windowed” (boolean): whether the game is in windowed mode.
- “windowWidth” and “windowHeight” (integers): the dimensions of the window.
- “musicVolume” and “fxVolume” (floats): the volumes between 0 and 1 of the music and the sound effects, respectively.

**Example:**

```
{
  "jumpKey": 62,
  "leftKey": 29,
  "rightKey": 32,
  "dropKey": 47,
  "growBranchButton": 0,
  "growBranchModKey": -1,
  "growLeafButton": 0,
  "growLeafModKey": 59,
  "resolutionIndex": 9,
  "fpsIndex": 4,
  "windowed": false,
  "windowWidth": 1280,
  "windowHeight": 720,
  "musicVolume": 1,
  "fxVolume": 1
}
```

## Level file

The level file will be saved in JSON format. It will contain information about the geometry of this level, including placement of resources and obstacles. The JSON file is an export from Tiled, and thus follows the Tiled specification. This section specifically describes the layers used on the tilemaps as well as custom properties for the tilemap. The layers are the following:

- “physics”: contains all the collidable tiles in the level, as well as other elements of the game geography including non-collidable balconies, clotheslines, and neon tutorial tiles.
- “resources”: contains all the water collection tiles.
- “hazards”: contains all the hazard-related tiles in the level, including bug hazard warnings and power lines.

The tilemap custom properties are:

- “background” (string): the asset code for the background file.
- “levelnumber” (integer): the level number.
- “time” (integer): the time on the timer at the start of the level, in seconds.
- “victory” (integer): the height of the finish line, in number of tiles.

**Example:**[3]

```
{
  "compressionlevel": -1,
  "height": 14,
  "infinite": false,
  "layers": [
    {
      "data": [...],
      "height": 14,
      "id": 1,
      "name": "physics",
      "opacity": 1,
      "type": "tilelayer",
      "visible": true,
      "width": 6,
      "x": 0,
      "y": 0
    },
    {
      "data": [...],
      "height": 14,
      "id": 2,
      "name": "resources",
      "opacity": 1,
      "type": "tilelayer",
      "visible": true,
      "width": 6,
      "x": 0,
      "y": 0
    },
    {
      "data": [...],
      "height": 14,
      "id": 3,
      "name": "hazards",
      "opacity": 1,
```

[3] The integer arrays corresponding to tiles in the “data” field of each layer have been omitted for brevity.

```
      "type": "tilelayer",
      "visible": true,
      "width": 6,
      "x": 0,
      "y": 0
    }
  ],
  "nextlayerid": 4,
  "nextobjectid": 1,
  "orientation": "orthogonal",
  "properties": [
    {
      "name": "background",
      "type": "string",
      "value": "gameplay:background1"
    },
    {
      "name": "levelnumber",
      "type": "int",
      "value": 4
    },
    {
      "name": "time",
      "type": "int",
      "value": 90
    },
    {
      "name": "victory",
      "type": "float",
      "value": 12
    }
  ],
  "renderorder": "right-down",
  "tiledversion": "1.10.2",
  "tileheight": 400,
  "tilesets": [
    {
      "firstgid": 1,
      "source": "tileset.tsx"
    },
    {
      "firstgid": 76,
      "source": "hazards.tsx"
    },
    {
      "firstgid": 82,
      "source": "rsrc.tsx"
    }
  ],
  "tilewidth": 600,
  "type": "map",
  "version": "1.10",
  "width": 6
}
```

### Tileset file

The tileset file will be saved in JSON format. It will contain information about the tileset for the tilemaps. The JSON file is an export from Tiled, and thus follows the Tiled specification. The game uses three tilesets corresponding to the three layers on the tilemap files. The physics tileset includes hitbox information for the tiles, as specified by the Tiled format. In addition, tiles in the hazards tileset include the custom property “type” (string, one of “powerline”, “bug”), representing the hazard type.

**Example:**

```
{
  "columns": 0,
  "grid": {
    "height": 1,
    "orientation": "orthogonal",
    "width": 1
  },
  "margin": 0,
  "name": "hazards",
  "spacing": 0,
  "tilecount": 6,
  "tiledversion": "1.10.2",
  "tileheight": 400,
  "tiles": [
    {
      "id": 0,
      "image": "powerline1.png",
      "imageheight": 400,
      "imagewidth": 600,
      "properties": [
        {
          "name": "type",
          "type": "string",
          "value": "powerline"
        }
      ]
    },
    {
      "id": 1,
      "image": "powerline2.png",
      "imageheight": 400,
      "imagewidth": 600,
      "properties": [
        {
          "name": "type",
          "type": "string",
          "value": "powerline"
        }
      ]
    },
    {
      "id": 2,
      "image": "powerline3.png",
      "imageheight": 400,
      "imagewidth": 600,
      "properties": [
        {
          "name": "type",
          "type": "string",
          "value": "powerline"
        }
      ]
    },
    {
      "id": 3,
      "image": "powerline4.png",
      "imageheight": 400,
      "imagewidth": 600,
      "properties": [
        {
          "name": "type",
          "type": "string",
          "value": "powerline"
        }
      ]
    },
    {
      "id": 4,
      "image": "bugzone-set1.png",
      "imageheight": 400,
      "imagewidth": 600,
      "properties": [
        {
          "name": "type",
          "type": "string",
          "value": "bug"
        }
      ]
    },
    {
      "id": 5,
      "image": "bugzone-set2.png",
      "imageheight": 400,
      "imagewidth": 600,
      "properties": [
        {
          "name": "type",
          "type": "string",
          "value": "bug"
        }
      ]
    }
  ],
  "tilewidth": 600,
  "type": "tileset",
  "version": "1.10"
}
```