Open Outdoor Games - Game Format Documentation

Overview

Open Outdoor Games uses a simple zip-based format for distributing GPS-based outdoor games. Each game consists of JavaScript code that defines interactive tasks, locations, and game logic, along with metadata and supporting assets.

Game Package Structure

A game package is a ZIP file containing:

```
game-package.zip
├── info.json          # Game metadata and configuration
├── game.js            # Main game logic and task definitions
└── <images>           # Game assets (optional) - e.g. cover.jpeg, trophy.png
```

Info.json (Game Metadata)

The `info.json` file contains essential game information displayed in the app details and library.

Game Info Fields

Field	Type	Description
`id`	string	Unique game identifier (reverse domain notation recommended)
`name`	string	Display name of the game
`description`	string	Brief description shown in the library

Field	Type	Description
`coverPhoto`	string	Filename of the cover image
`startLocation`	object	Starting location configuration
`finishLocation`	object	Ending location configuration
`attributes`	object	Game characteristics and difficulty ratings

Note: the attributes field is optional but highly recommended to include.

Location Object Structure

Location configuration for start and finish points. The game is allowed to start only when the user is close to the start location (75m radius).
If the game is not tied to specific locations, do not put coordinates, just a text description.

Example:

```
{
  "text": "Human-readable location description",
  "coordinates": {
    "lat": 50.0971869,
    "lng": 14.4038831
  }
}
```

Attributes Object Structure

Object containing key-value pairs describing game characteristics. Usually includes expected duration, distance or physical/mental difficulty ratings.

Example:

```
{
  "Atrribute 1 Name": "Attribute 1 Value",
  "Atrribute 2 Name": "Attribute 2 Value"
}
```

Example info.json

```
{
  "id": "com.rejnek.dejvice.alpha",
  "name": "Dejvice Game",
  "description": "A game that takes you through interesting places in Dejvice and surround
  "coverPhoto": "cover.jpeg",
  "startLocation": {
    "text": "Hradčanská tram stop, Prague 6",
    "coordinates": {
      "lat": 50.0971869,
      "lng": 14.4038831
    }
  },
  "finishLocation": {
    "text": "Kaufland Dejvice, Prague 6",
    "coordinates": {
      "lat": 50.1118728,
      "lng": 14.3926511
    }
  },
  "attributes": {
    "Expected duration": "1 hour",
    "Mental difficulty": "2/5",
    "Physical difficulty": "2/5"
  }
}
```

Game.js (Game Logic)

The game logic is written in JavaScript and defines tasks, user interactions, and game flow.

The file consist of game variables and task definitions.

The first task needs to be named `start` as it is the entry point of the game.

Game variables

Global variables needs to be prefixed with `_` (e.g. `_score`, `_timerStart`) and are accessible across all tasks.
The variable values are saved if the game is paused or the app is closed.

Example:

```
var _score = 0;                 // Player score
var _timerStart = Date.now();   // Game start time
```

Task Structure

Each task is a JavaScript object with lifecycle methods. The name needs to be unique and is used to reference the task.
The name of the first task must be `start`.

```
const taskName = {
    // Optional: called only the first time the task is started. Suitable for variable inc
    onStartFirst: () => {
        // First-time setup logic
    },

    // Called every time task becomes active
    onStart: () => {
        // Task content and interactions
        // never put variable increase here - it will increase every time when the task is
    },

    // Optional: location boundaries for location-based tasks. Defined as polygon of [lat,
    loc: [
        [lat1, lng1],
        [lat2, lng2],
        [lat3, lng3]
    ]
}
```

Location-Based Tasks

Tasks can be tied to specific geographic locations. In such case, the `onStart` and `onStartFirst` respectively are triggered when the user is in inside the defined area.
If you want to hide the task by some condition, you can use `disable("taskName")` and `enable("taskName")` functions that will prevent or allow location detection for selected task.

```
const locationTask = {
    loc: [
        [50.106943, 14.394933],
        [50.107247, 14.394844],
        [50.107293, 14.395915],
        [50.106949, 14.396097]
    ],
    onStartFirst: () => {
        // Triggered *only once* on first visit, suitable for updating variables
        _score += 5;
    },
    onStart: () => {
        // Triggered *every* time user enters area
        heading("You've arrived!");
        // Task content...
    }
}
```

Game API Functions

The game engine provides various functions for creating interactive content:

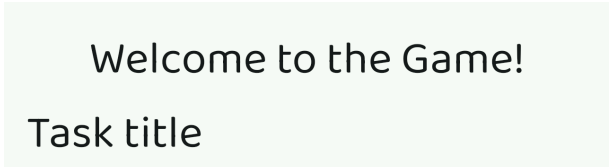
UI Elements

heading(text, alignment)

Display title text with optional alignment (`"left"` , `"center"` , `"right"`).

Example:

```
heading("Welcome to the Game!", "center");
heading("Task title"); // defaults to left alignment
```



text(content)

Display paragraph text.

Example:

```
text("This is an example of game instructions or story text.");
text("You can include newline characters \n for formatting or put multiple text() calls be
```

This is an example of game instructions or story text.
You can include newline characters
for formatting or put multiple text() calls below each other.

image(filename)

Display an image from the game assets.

Example:

```
image("trophy.png");
image("./images/illustration.jpeg"); // relative path also works
```

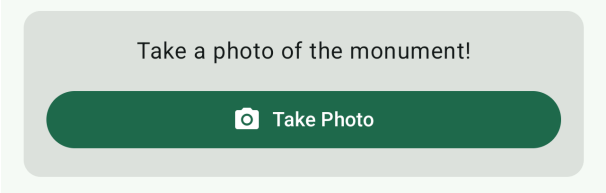
takePicture(prompt)

Prompt user to take a photo with their device camera.
The photo is saved to the game storage and can be viewed later with `showAllImages()` function.

Note: Current limitation is one photo per task. Taking a new photo in the same task overwrites the previous one.

Example:

```
takePicture("Take a photo of the monument!");
```



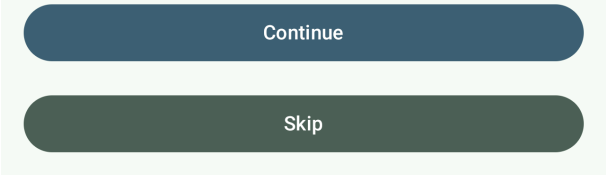
User interaction

button(label, callback)

Create an interactive button that executes a callback function when pressed.

Example:

```
button("Continue", () => showTask("nextTask"));
button("Skip", () => {
    _score -= 5;
    popUp("Task skipped", "otherTask");
});
```

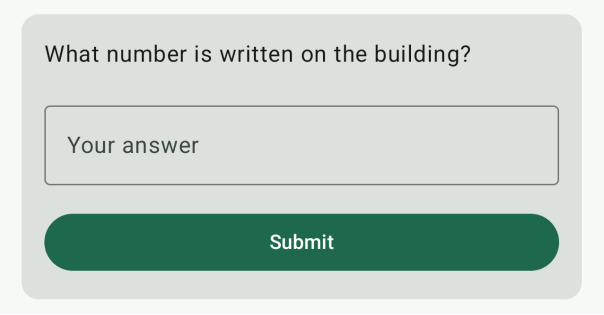


question(prompt, callback)

Display an open question prompt and handle user answer via callback.

Example:

```
question("What number is written on the building?", (answer) => {
    if( answer === "42") {
        _score += 10;
        popUp("Correct!", "nextTask");
    } else {
        popUp("Wrong answer, try again.", "currentTask");
    }
});
```

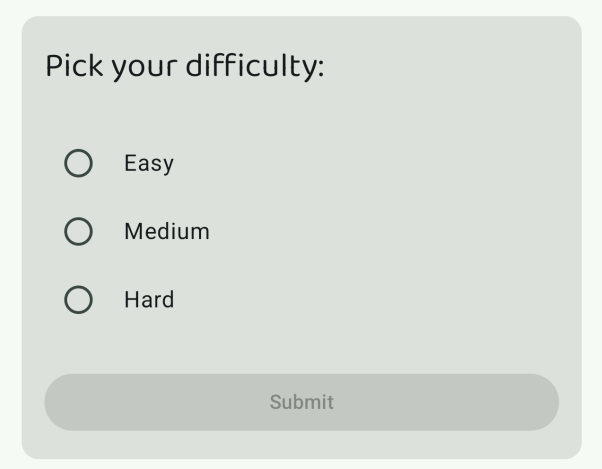


multichoice(prompt, callback, ...options)

Present multiple choice question with predefined options. The number of selected option is passed to the callback.
Keep in mind that the numbering starts from 0!

Example:

```
multichoice("Pick your difficulty:", (choice) => {
    if (choice === 0) {
        _difficulty = "Easy";
    } else if (choice === 1) {
        _difficulty = "Medium";
    } else {
        _difficulty = "Hard";
    }
    showTask("gameStart");
}, "Easy", "Medium", "Hard");
```

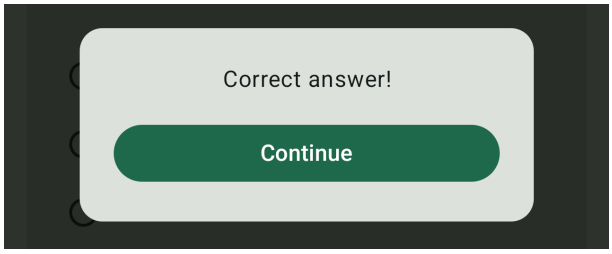


popUp(message, nextTask)

Show a popup message and automatically continue to the next task. This function is meant to be used from callbacks after user action.
You can omit the `nextTask` parameter to just show a message without changing the current task.

Example:

```
popUp("Correct answer!", "nextTask");
popUp("The hint is 1234."); // stays in current task
```



Game Flow Control

showTask(taskName)

Open another task by its name. The current task is closed and the onStart of the new task is executed.

Example:

```
showTask("nextChallenge");
```

enable(taskName)

Enable a location-based task (makes it active for location detection). This has no effect for showing the task by `showTask` command.

Note: All locations are enabled by default.

Example:

```
enable("secretLocation");
```

disable(taskName)

Disable a location-based task (prevents further location detection). This has no effect for showing the task by `showTask` command.

Example:

```
disable("completedTask");
```

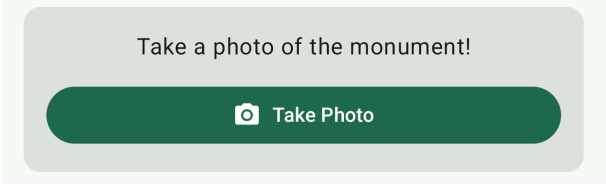
Navigation UI elements

distance(lat, lng)

Display the distance from current location to specified coordinates.

Example:

```
distance(50.1094, 14.3933);
```



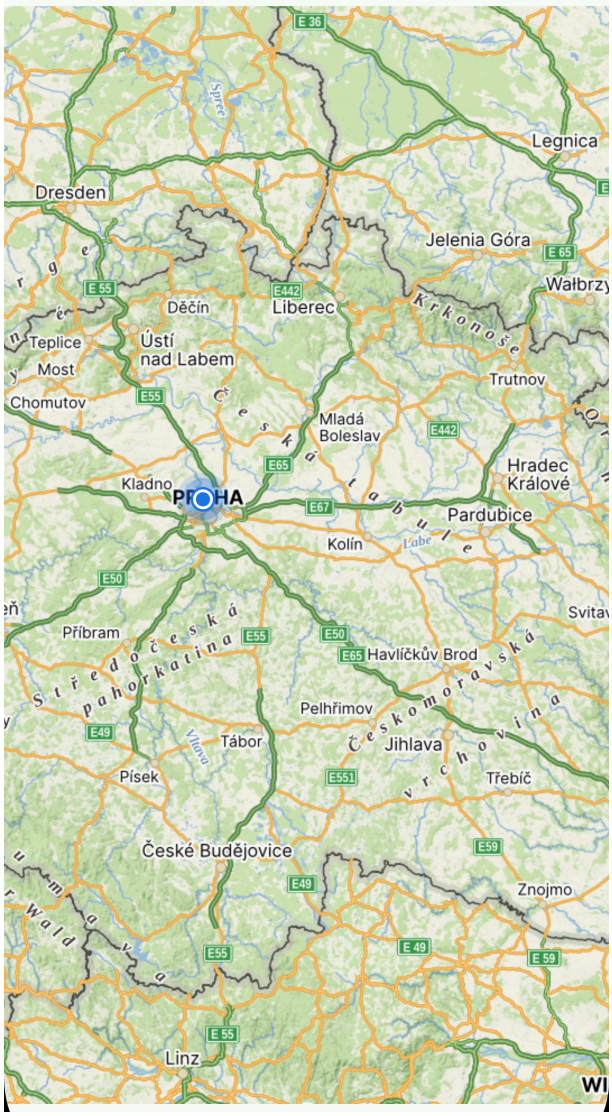
simpleMap(image, topLeftLat, topLeftLng, bottomRightLat, bottomRighLng)

Display a map overlay image with geographic bounds. The app will show user's current location on the map as a blue dot.

- The image needs to be part of the game assets, the same way as images used in `image()` function.
- The coordinates are for the top-left and bottom-right corners of the image.

Example:

```
simpleMap("map.png", 51.6620556, 13.2027478, 48.2060719, 16.3777964);
```



Finish Game

board(title, ...pairs)

Display a score board with title and key-value pairs.

Example:

```
board("Final Results",
      "Score", _score,
      "Time", Math.floor((Date.now() - _timerStart) / 60000) + " min",
      "Difficulty", "Hard");
```

showAllImages(title)

Display a gallery of all photos captured during the game.

Example:

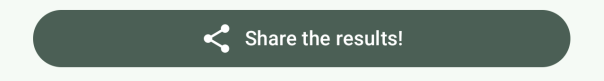
```
showAllImages("Memories from the Game");
```

shareButton()

Add a social sharing button that shares the current screen. Most useful on the final results screen.

Example:

```
shareButton();
// Allows users to share their progress or results
```

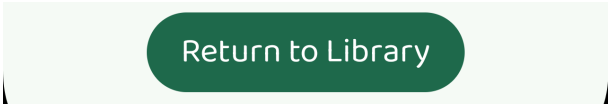


finishGameButton(label)

Create a button that returns to the library and marks the game as finished.

Example:

```
finishGameButton("Return to Library");
```



Debug Functions

debugPrint(message)

Log debug messages (visible in development/testing mode).

Example:

```
debugPrint("Player reached checkpoint: " + _currentLocation);
```

Example Game Structure

```
// Global game state
var _score = 0;
var _timerStart = Date.now();
var _timeElapsed = 0;

// Starting task
const start = {
    onStartFirst: () => {
        _timerStart = Date.now();
    },

    onStart: () => {
        heading("Game Begins!");
        text("Welcome to the adventure.");
        text("You need to go to the first challenge location.");
        button("Skip", () => {
            showTask("firstChallenge");
        });
    }
}

// Location-based task
const firstChallenge = {

    // The locaiton is defined as a polygon of [lat, lng] pairs - minimun number is 3
    loc: [
        [50.106943, 14.394933],
        [50.107247, 14.394844],
        [50.107293, 14.395915]
    ],

    // The onStart activates when the user visit the locaiton OR when the task is shown by
    onStart: () => {
        heading("First Challenge");
        question("What do you see?", (answer) => {
            if (answer.toLowerCase().includes("statue")) {
                _score += 10;
                popUp("Correct!", "finish"); // move to finish task
            } else {
```

```
                popUp("Try again!"); // stays in the same task
            }
        });
    }
}

// Final task
const finish = {
    onStartFirst: () => {
        // The time elapsed needs to be calculated once and stored in the global variable
        // this way, it won't change if user revisits the finish task
        // the same principle applies to score or any other variable you want to freeze/up
        _timeElapsed = Math.floor((Date.now() - _timerStart) / 60000);
    },
    onStart: () => {
        disable("firstChallenge");
        heading("Congratulations!");
        board("Results", "Score", _score, "Time", _timeElapsed + " min");
        finishGameButton("Return to Library");
    }
}
```