Battleship Game
Play instructions via the API

Index

Models, Fields and Value Ranges

```
Game:
  string   "title"                    (Must be unique among all games)
  integer  "player_1_id"              (The Player ID for player 1)
  integer  "player_2_id"              (The Player ID for player 2)
  integer  "whose_move"               (0 == waiting for player_1 to move,
                                       1 == waiting for player_2)
  integer  "move_counter"             (Starts at 0, incremented each move.)
  integer  "player_1_fleet_status"    (Each bit represents a ship part,
                                       1 == healthy, 0 == hit)
  integer  "player_2_fleet_status"    (ship parts order is described below.)
  text     "player_1_fleet_coords"    (Array of ship parts coordinates, i.e.
  text     "player_2_fleet_coords"     ["H10",
                                        "D4", "D5",
                                        "A7", "B7", "C7",
                                        "H1", "H2", "H3", "H4",
                                        "F4", "F5", "F6", "F7", "F8"])
  datetime "created_at"               (Managed by the database)
  datetime "updated_at"               (Managed by the database)

Move:
  integer  "game_id"                  (ID of Game to which this move applies)
  integer  "player_id"                (ID of Player making this move)
  string   "attack_coords"            (Player's target coordinate, i.e. “E7”)
  integer  "move_number"              (Set by server when a move is created)
  string   "ship_part_hit"            (name of the ship hit, if any)
  boolean  "hit"                      (true if the attack hit)
  boolean  "ship_sunk"                (true if the attack sunk a ship)
  boolean  "fleet_sunk"               (true if the defender's fleet is sunk)
  text     "message"                  (summarizes the move and its result)
  datetime "created_at"               (Managed by the database)
  datetime "updated_at"               (Managed by the database)

Player:
  string   "name"                     (Must be unique among all players)
  datetime "created_at"               (Managed by the database)
  datetime "updated_at"               (Managed by the database)
```

Explanation of the Fleet Coordinates and Status

There are several version of the game of battleship, including variations in the ship lengths. This game follows the battleship version which has five ships of lengths 1, 2, 3, 4, & 5.

Each ship has as many parts as its length. For example, the Cruiser has a length of 3 so it has parts: Cruiser_1, Cruiser_2, Cruiser_3. The coordinates of the parts of a ship must be adjacent and arranged in a straight line, vertically or horizontally, but not diagonally.

Each player's fleet coordinates array elements are composed of the player's ship parts and those parts are ordered according to a consistent ship order, as follows for indices 0 through 14 of the array:
  Submarine_1,
  Destroyer_1, Destroyer_2,
  Cruiser_1, Cruiser_2, Cruiser_3,
  Battleship_1, Battleship_2, Battleship_3, Battleship_4,
  Aircraft_Carrier_1, Aircraft_Carrier_2, Aircraft_Carrier_3, Aircraft_Carrier_4, Aircraft_Carrier_5,

Similarly, the status of a player's fleet is represented by an integer in which each bit corresponds to a specific ship part, and the same ordering from the fleet coordinates applies to the fleet_status bits, considered from the least to the most significant bit. So bit 0b001 corresponds to the Submarine, and bits 0b110 correspond to the Destroyer, and so forth. If the bit is 1 then the corresponding ship part is healthy, while a value of zero means that part of the ship has been hit. When all the bits of a ship are zero, the ship is sunk and when a player's fleet status integer is zero, then their fleet is sunk.

Setting Up and Playing a Game

API Request Headers

Any of the non-GET API requests (create/POST, update/PATCH) should include in their Header the parameter “`Content-Type`” with value “`application/json`”, to indicate that the content of the POST or PATCH is in JSON format. Commands may also include the “`Accept`” header, again with “`application/json`” as the value, though all the commands will reply with JSON anyway.

Creating Players

In order to play a game, two players will first need to be created using the create player command.

Request

Method	URL
`POST`	`api/v1/players`

Type	Params	Values
`HEAD`	`Content-Type`	`“application/json”`
`POST`	`{ "player": { "name": <player_name> } }`	`string`

`name`
Specifies the player's name and must be unique amongst all players. If a request is submitted without a name or one that is already in use, an error code will be returned.

Response

Status	Response
`201`	`{ "id": 2, "name": "Jenny", "created_at": "2016-07-13T23:06:04.528Z", "updated_at": "2016-07-13T23:06:04.528Z"}`
`422`	`{"error":"param is missing or the value is empty: player"}`
`422`	`{"error":"param is missing or the value is empty: name"}`
`422`	`{"error":"Validation failed: Name has already been taken"}`

Getting Defaults for a New Game

Before creating a game, a client can optionally first issue the new game command **(GET <battleship_server_url>/api/v1/games/new)** which will return a JSON object which includes the following new game initialization values:

```
{
  "whose_move":             0,     # 0 means player_1 moves first
  "move_counter":           0,     # 0 means 0 moves have been made
  "player_1_fleet_status": 32767, # 32767 is the value of bits 0 through 15
  "player_2_fleet_status": 32767, # all set to 1, i.e. an intact fleet
  "player_1_fleet_coords": ["A1", "B3", "C3", …]
                                   # an array of 15 coordinate string values,
                                   # for the 15 ship parts
  "player_2_fleet_coords": ["H4", "F1", "F2", …]
                                   # As above, a valid random arrangement of
                                   # ships on the 10x10 board.
}
```

Creating A Game

The create game command **(POST <battleship_server_url>/api/v1/games)** can now be issued, providing in the POST body the values from the aforementioned new game command and adding in values for title and player Ids for players 1 & 2:

```
  "title": "Battle of Yamen",      # Must be unique among all games.
  "player_1_id": <integer>,        # IDs optional, but if not supplied, will
  "player_2_id": <integer>,        # need to update game with IDs to play.
```

To handle the possibility that the two players will join the game at different times, `player_x_id` and its related `player_x_fleet_status` and `player_x_fleet_coords` are optional parameters when creating a game. However, the fleet coordinates and status are required if the corresponding player ID is provided. If a game is created but `player_1_id` and/or `player_2_id` is null, then the game state will be one of:

```
  STATE_WAITING_ALL_PLAYERS_TO_JOIN
  STATE_WAITING_PLAYER_1_TO_JOIN
  STATE_WAITING_PLAYER_2_TO_JOIN
```

Request

Method	URL
POST	api/v1/games

Type	**Params**	**Values**
`HEAD`	`Content-Type`	`“application/json”`
`POST`	`title`	`string`
`POST`	`player_1_id`	`integer`
`POST`	`player_2_id`	`integer`
`POST`	`whose_move`	`0|1`
`POST`	`move_counter`	`integer`
`POST`	`player_1_fleet_status`	`integer`
`POST`	`player_2_fleet_status`	`integer`
`POST`	`player_1_fleet_coords`	`array of strings`
`POST`	`player_2_fleet_coords`	`array of strings`

`title`
The title or name of the game. Must be unique amongst all games and will result in an error response if a game is created or updated using an existing title.

`whose_move` `[optional]*`
Value indicating whose turn it is, that is which player will next be allowed to submit a new attack coordinate. A value of 0 indicates player_1, while 1 means player_2. This should be set to 0 for a newly created game, to ensure player_1 will get the first move and player_2 will get the last.

`move_counter` `[optional]*`
This is the count of moves which have been played in this game, and should be set to 0 for a newly created game.

`player_1_id` `[optional]*`
The Player ID for player_1, who by default plays first (but not last). The player_x_id's are optional and do not need to be provided at time of game creation, but they need to both be present for the game to accept moves. A player can join a game with the update command.

`player_2_id` `[optional]*`
The Player ID for player_2, who by default plays second (and gets the last move of the game).

`player_1_fleet_status` `[optional]*`
The health of player 1's fleet is represented by this integer, in which each bit corresponds to a specific part of one ship, where a 1 means that part has not yet been hit and a 0 means that part

has been bit. If all parts of a ship are hit, the ship is sunk. In the current version of this game, there are 5 ships with lengths 1, 2, 3, 4, and 5, thus there are a total of 15 ship parts (1+2+3+4+5=15) and a healthy fleet is indicated as binary 0b111111111111111 = 32767. Higher order bits beyond bit position 15 are ignored.

`player_2_fleet_status` `[optional]*`
The health of player 2's fleet is represented by this integer, where 32767 means an intact fleet.

`player_1_fleet_coords` `[optional]*`
The coordinates of the ships comprising player 1's fleet are represented in this array of coordinate strings. Coordinates should be specified as a letter and number combination, where the letters range from A-H and the numbers from 1-10 (for a 10x10 grid), e.g. “A10” or “F5”. (Valid random fleet layouts can be obtained by making GET requests to api/v1/games/new). The coordinates of ship parts in the fleet coordinates array should be in the following order: [submarine_1, destroyer_1, 2, cruiser_1, 2, 3, battleship_1, 2, 3, 4, aircraft_carrier_1, 2, 3, 4, 5]. the coordinates for the parts of a ship must be contiguous and arranged vertically or horizontally and not diagonally. An example of a valid fleet layout:
["H2",   "I2", "I3",   "D10", "E10", "F10",   "C7", "D7", "E7", "F7",   "A4", "A5", "A6", "A7", "A8"]

`player_2_fleet_coords` `[optional]*`
The coordinates of the ships comprising player 2's fleet are represented in this array of coordinate strings.

`*NOTE on optional values:`
While most fields are optional, sets of them become mandatary when one or both player_ids are supplied. Specifically, if player_x_id is present (where x == 1 or 2}, then player_x_fleet_coords and player_x_fleet_status become required and get validated. If both player_ids are present, then whose_move and move_counter are both required.

Response

Status	Response

201	**A JSON object representation of the created game is given, for example:**
	`{`
	`  "id": 3,`
	`  "title": “Battle of Jutland”,`
	`  "player_1_id": 1,`
	`  "player_2_id": 2,`
	`  "whose_move": 0,`
	`  "move_counter": 0,`

	"player_1_fleet_status": 32767, "player_2_fleet_status": 32767, "player_1_fleet_coords": ["I3", "H10", "I10", "E3", "F3", "G3", "A9", "B9", "C9", "D9", "B4", "B5", "B6", "B7", "B8"], "player_2_fleet_coords": ["F8", "I1", "I2", "C9", "D9", "E9", "D3", "E3", "F3", "G3", "A2", "A3", "A4", "A5", "A6"], "created_at": “2016-07-14T16:10:14.588Z”, "updated_at": “2016-07-14T16:10:14.588Z” }
422	{"error": "param is missing or the value is empty: game"}
422	{"error": "Validation failed: Title can't be blank"}
422	{"error": "Validation failed: Player 1 fleet status must be less than or equal to 32767"}
422	{"error": "Validation failed: Player 1 fleet status must be greater than or equal to 0"}
422	{"error": "Validation failed: Player 1 fleet coords must have 15 unique coordinate values"}
422	{"error": "Validation failed: Player 1 fleet coords has these 2 elements in the wrong format: ["N3", "10H"] (values should range from A1 to J10)"}
422	{"error": "Whose move must be 0 or 1"}
422	{"error": "Move counter is not a number"}
422	{"error": "Move counter must be greater than or equal to 0"}

500	`{"error":"Something went wrong. Please try again later."}`

Updating A Game

If the game state is waiting for one or both players to join, the missing player(s) will need to join the game using the update game command **(PATCH <battleship_server_url>/api/v1/games/<game_id>)**. If, for instance, player_2_id is missing from the game the following would be required in the update command's POST body:

```
{  "game": { "player_1_id": 1 } }
```

Getting the Game Status

A player can not submit a move if it is not their turn to play. This prevents a player from issuing a bunch of firing commands and winning even before the other player gets a chance to move. In order to determine whether it is their turn to play, a game client can periodically issue a game status command **(GET <battleship_server_url>/api/v1/games/<game_id>/status)**. The response will include the current game state and, if applicable, the last move and result, and whose turn it is now. An example response may look like this:

```
{
  "id": 3,
  "title": "Battle of Jutland",
  "player_1_id": 1,
  "player_2_id": 2,
  "whose_move": 1,
  "move_counter": 7,
  "created_at": "2016-07-15T23:18:26.700Z",
  "updated_at": "2016-07-18T06:40:26.732Z",
  "game_state": 4,
  "game_state_message": "Waiting for player 2 to move",
  "last_move": {
     "id": 20,
     "game_id": 3,
     "player_id": 1,
     "move_number": 7,
     "attack_coords": "H10",
     "ship_part_hit": null,
     "hit": false,
     "ship_sunk": false,
     "fleet_sunk": false,
     "message": "Cedric targeted: H10 and missed.",
     "created_at": "2016-07-18T06:40:26.623Z",
     "updated_at": "2016-07-18T06:40:26.623Z"
  }
}
```

Or the response may look like this if the game is still waiting for one of the players to join:

```
{
  "id": 2,
  "title": "game too",
  "player_1_id": 1,
  "player_2_id": null,
  "whose_move": 0,
  "move_counter": 0,
  "created_at": "2016-07-07T05:48:24.712Z",
  "updated_at": "2016-07-07T06:23:38.556Z",
  "game_state": 2,
  "game_state_message": "Waiting for player 2 to join",
  "last_move": {}
}
```

Playing A Game

To make a move in the battleship game, a client/player should issue a create move command. The server will return an error if the player is attempting to play out of turn or after the game is over. Otherwise, it will calculate the effects of the move and save these to the move and game.

Request

Method	URL
POST	`api/v1/games/<game_id>/moves`

Type	Params	Values
HEAD	Content-Type	“application/json”

POST	`{`
	`  "move": {`
	`    "game_id": 3,`
	`    "player_id": 1,`
	`    "attack_coords": "A1"`
	`  }`
	`}`
		
		integer
		integer
		string

`game_id`
Specifies the ID of the game upon which this move is to be played.

`player_id`
Specifies the ID of the player who is making this move (attacking the other player's fleet).

`attack_coords`
Specifies the grid coordinate to attack, in the format “<letter><number>”, where <letter> should be in the range of “A” through “J” and <number> should be in the range 1-10. (However, if the coordinates are specified as <number><letter>, the server will transpose the coordinates before trying the attack.)

Response

Status	Response
`201`	**Possible response when the attack missed:** `{ "id": 15, "game_id": 3, "player_id": 1, "move_number": 1, "attack_coords": "A1", "ship_part_hit": null, "hit": false, "ship_sunk": false, "fleet_sunk": false, "message": "Cedric targeted: A1 and missed.", "created_at": "2016-07-18T01:52:11.205Z", "updated_at": "2016-07-18T01:52:11.205Z" }`
`201`	**Possible response when the attack hit a ship but did not sink it:** `{ "id": 16, "game_id": 3, "player_id": 2, "move_number": 2, "attack_coords": "H10", "ship_part_hit": "Destroyer", "hit": true,`

```
  "ship_sunk": false,
  "fleet_sunk": false,
  "message": "Jenny targeted: H10 and hit Cedric's
Destroyer!",
  "created_at": "2016-07-18T01:53:11.205Z",
  "updated_at": "2016-07-18T01:53:11.205Z"
}
```

201

Possible response when the attack hit and sunk a ship:

```
{
  "id": 15,
  "game_id": 3,
  "player_id": 1,
  "move_number": 3,
  "attack_coords": "F8",
  "ship_part_hit": "Submarine",
  "hit": true,
  "ship_sunk": true,
  "fleet_sunk": false,
  "message": "Cedric targeted: F8 and sunk Jenny's
Submarine!",
  "created_at": "2016-07-18T01:54:11.205Z",
  "updated_at": "2016-07-18T01:54:11.205Z"
}
```

201

Possible response when the attack hit a ship and sunk the fleet:

```
{
  "id": 16,
  "game_id": 3,
  "player_id": 2,
  "move_number": 42,
  "attack_coords": "B4",
  "ship_part_hit": "Destroyer",
  "hit": true,
  "ship_sunk": true,
  "fleet_sunk": true,
```

	"message": "Jenny targeted: H10 and hit Cedric's Air Craft Carrier! And destroyed Cedric's fleet!", "created_at": "2016-07-18T02:34:11.205Z", "updated_at": "2016-07-18T02:34:11.205Z" }
404	{"error": "Couldn't find Game with 'id'=20"}
422	{"error":"param is missing or the value is empty: move"}
422	{"error":"param is missing or the value is empty: game_id"}
422	{"error":"param is missing or the value is empty: player_id"}
422	{"error":"param is missing or the value is empty: attack_coords"}
423	{"error":["Player ID (1)) does not match whose turn it is (2)"]}
500	{"error":"Something went wrong. Please try again later."}

Winning A Game

Even though each player's moves increments the game's move_counter, it can be considered as if both players are firing simultaneously. Therefore, if player_1 sinks player_2's fleet, since player_1 moved first player_2 is given one last turn, as if they had both fired at the same time. Thus it is possible for both players to sink each others' fleets resulting in a tie game. Otherwise, only one player will win. The aforementioned game status command can be issued to confirm the final disposition of the game. Moves can no longer be created once a game has been won.