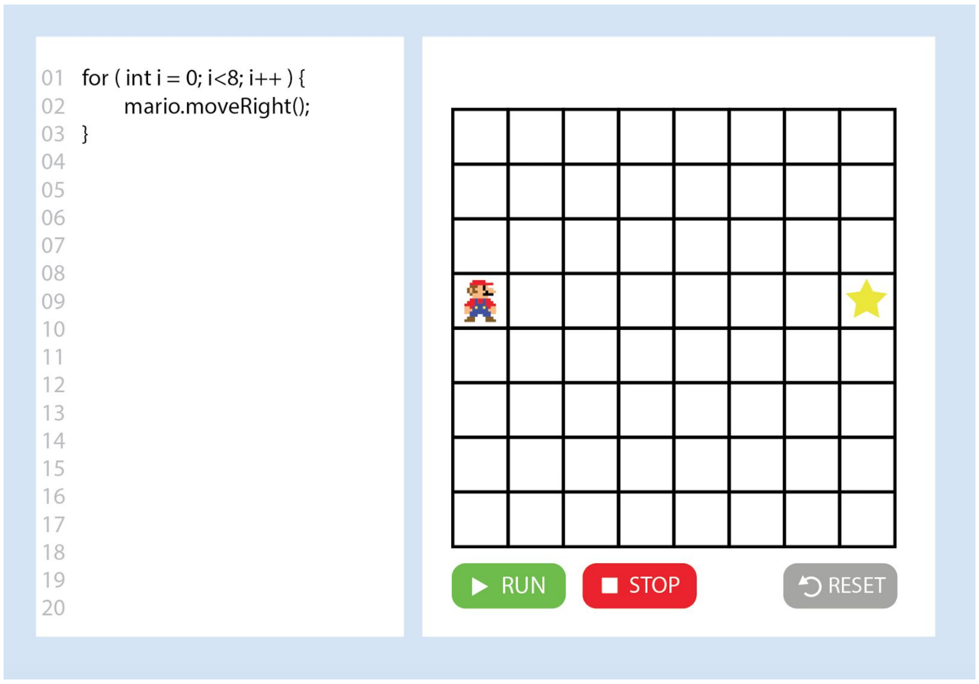
Sprint Reports and Design Design Outcomes

Initial mockup of MVP (playing a level) from project plan:



```
for ( int i = 0; i<8; i++ ) {
      mario.moveRight();
}
```

Sprint 1

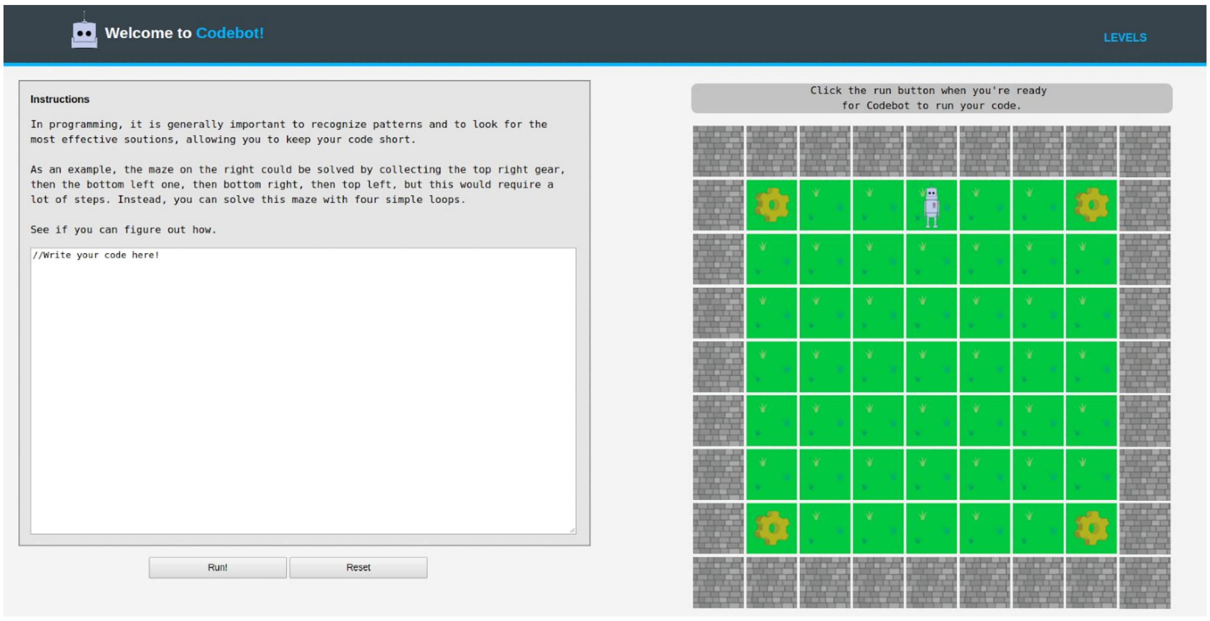
- Server will pass levels via template (at least, for now). We decide to guard against users finding out all levels (by trying URLs), but not against them finding the JS-encoding of a level.
- Server-side indirection: the server should have its own representation of the World which it then passes (e.g. via template) to the JS in an agreed format, which the JS then converts into its own representation. For now / MVP, the server will just directly pass a level description in raw format, with the knowledge of how the JS interprets it.
- Look into using jQuery and lodash as our basic libraries - vanilla javascript can't express selectors very well
- Improve the basic layout of the webpage

Sprint 2

- Decided that having functionality for teachers and classes is out of scope and not very important. Instead, we'll focus on making the user experience great for primary school to early high school kids - our target audience.
  - This involves creation of child-friendly levels with fun and simple gameplay rather than user accounts with a lot of potentially unnecessary features
  - Classroom and teacher features may be omitted later on in favour of better level features

Sprint 3

- Adjust the scope: at least for the moment users can play all levels uninhibited.
- Current UI as of demo:



Instructions

In programming, it is generally important to recognize patterns and to look for the most effective soutions, allowing you to keep your code short.

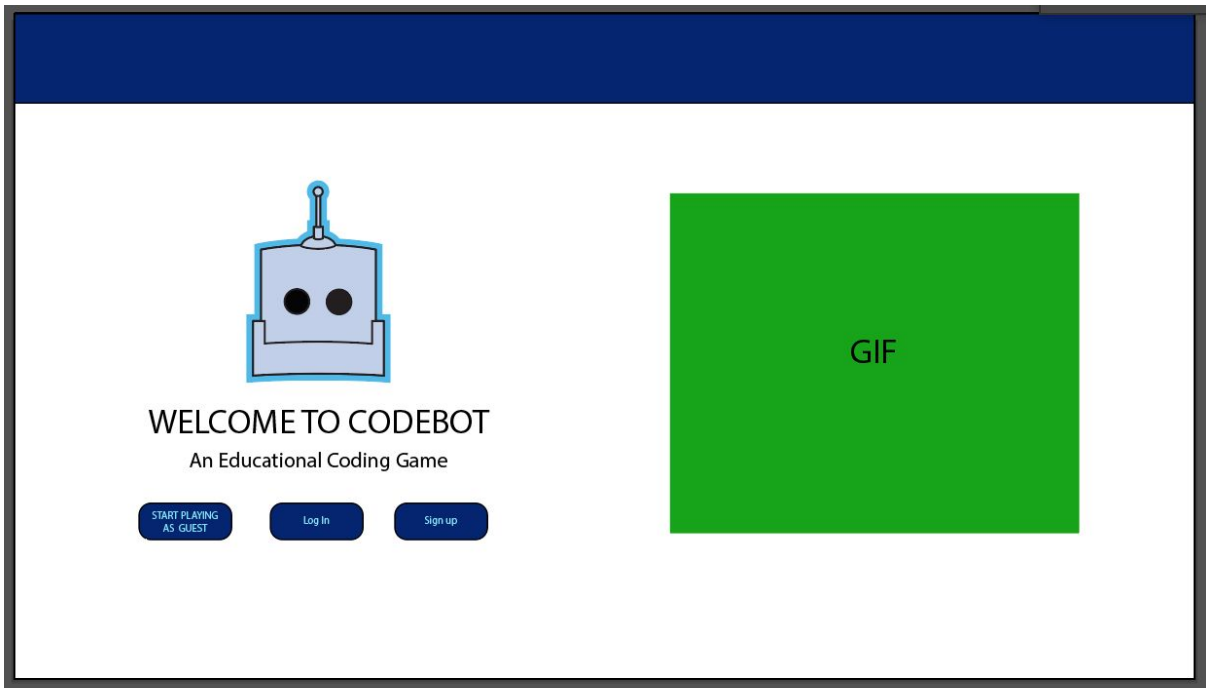
As an example, the maze on the right could be solved by collecting the top right gear, then the bottom left one, then bottom right, then top left, but this would require a lot of steps. Instead, you can solve this maze with four simple loops.

See if you can figure out how.

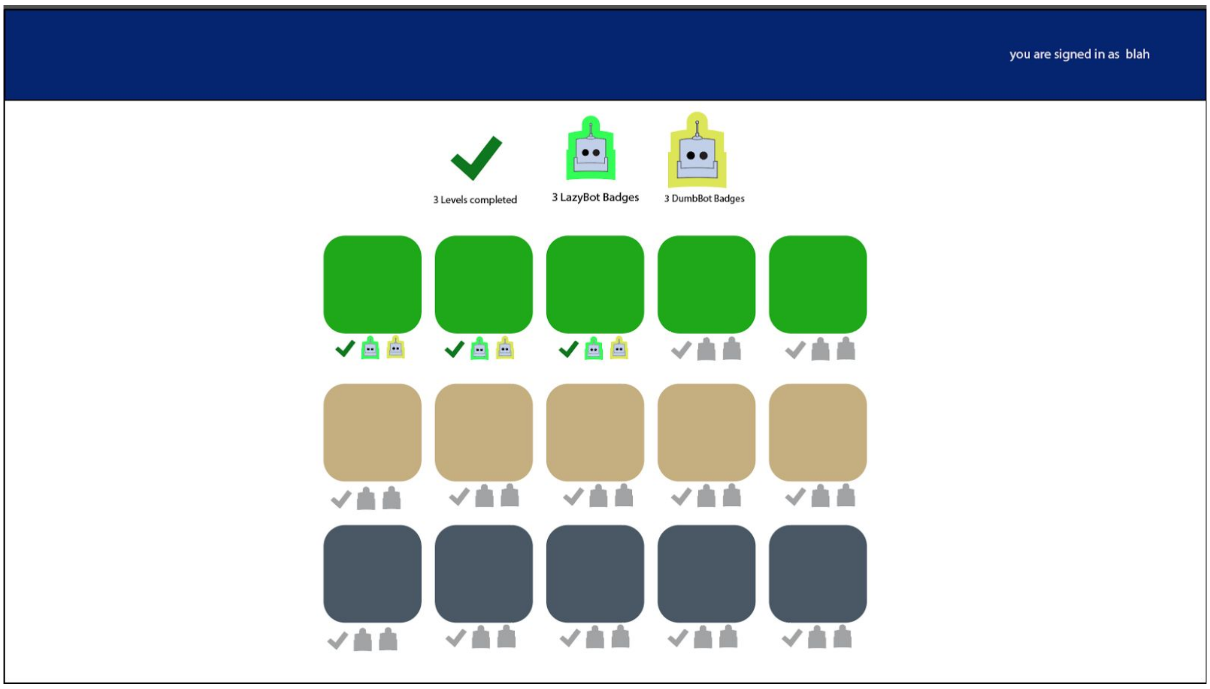
```
//Write your code here!
```

Click the run button when you're ready for Codebot to run your code.

- Mockup of intended future landing page



- Mockup of intended future level selection page



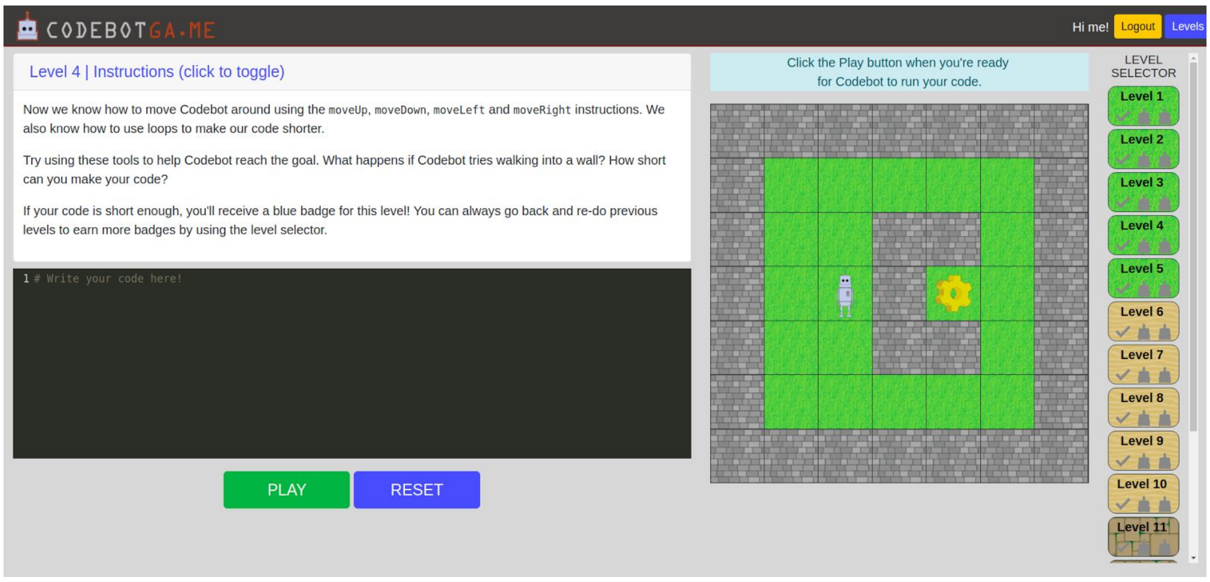
Sprint 4

- Change language used in level instructions to be age-appropriate for target audience (primary school students).
- For final product, won't protect against users who go out of their way to game the system / mark levels as done which they haven't already
    - Ideally this would be done by re-running their code server-side but if this game is targeted at primary school kids who can't already code, this likely won't be a problem.
    - Since the game is made for their learning, they would only be cheating themselves.

Sprint 5

- Landing page mockups
- Handling logins: choosing to use simpler username / password (store password hash). Initially no password recovery.
- Use a codebox with syntax highlighting so students can see which terms are recognised / correct as appropriate
- REST API at `/update_score` for updating scores [POST]. If optional fields are left out, they will not be updated. Note that user id is passed implicitly from the session (the endpoint requires being logged in). Accepts the following in a JSON object.
    - `codename` : the codename of the level. If no such level exists, will return `404`.
    - `code_score` : (optional) current score assigned to the code
    - `execution_score` : (optional) execution score assigned to the code
    - Returns `200 OK` if successful as well as some extra information in a JSON object:
        - "`code_score`": the new code_score for this (user, level) pair. If none exists, defaults to 2^30.

        - "`execution_score`": the new execution_score for this (user, level) pair. If none exists, defaults to 2^30.
        - "`level`": codename (e.g. `level7`) of the level that was just updated
        - "`level_id`": database id of the level that was just updated
        - "`success`": `True` if the operation was successful, or `False` otherwise. In particular, if the user is not logged in, returns `False`.
- Use a level selector instead of a next level button



Level 4 | Instructions (click to toggle)

Now we know how to move Codebot around using the `moveUp`, `moveDown`, `moveLeft` and `moveRight` instructions. We also know how to use loops to make our code shorter.

Try using these tools to help Codebot reach the goal. What happens if Codebot tries walking into a wall? How short can you make your code?

If your code is short enough, you'll receive a blue badge for this level! You can always go back and re-do previous levels to earn more badges by using the level selector.

```
# Write your code here!
```

Click the Play button when you're ready for Codebot to run your code.