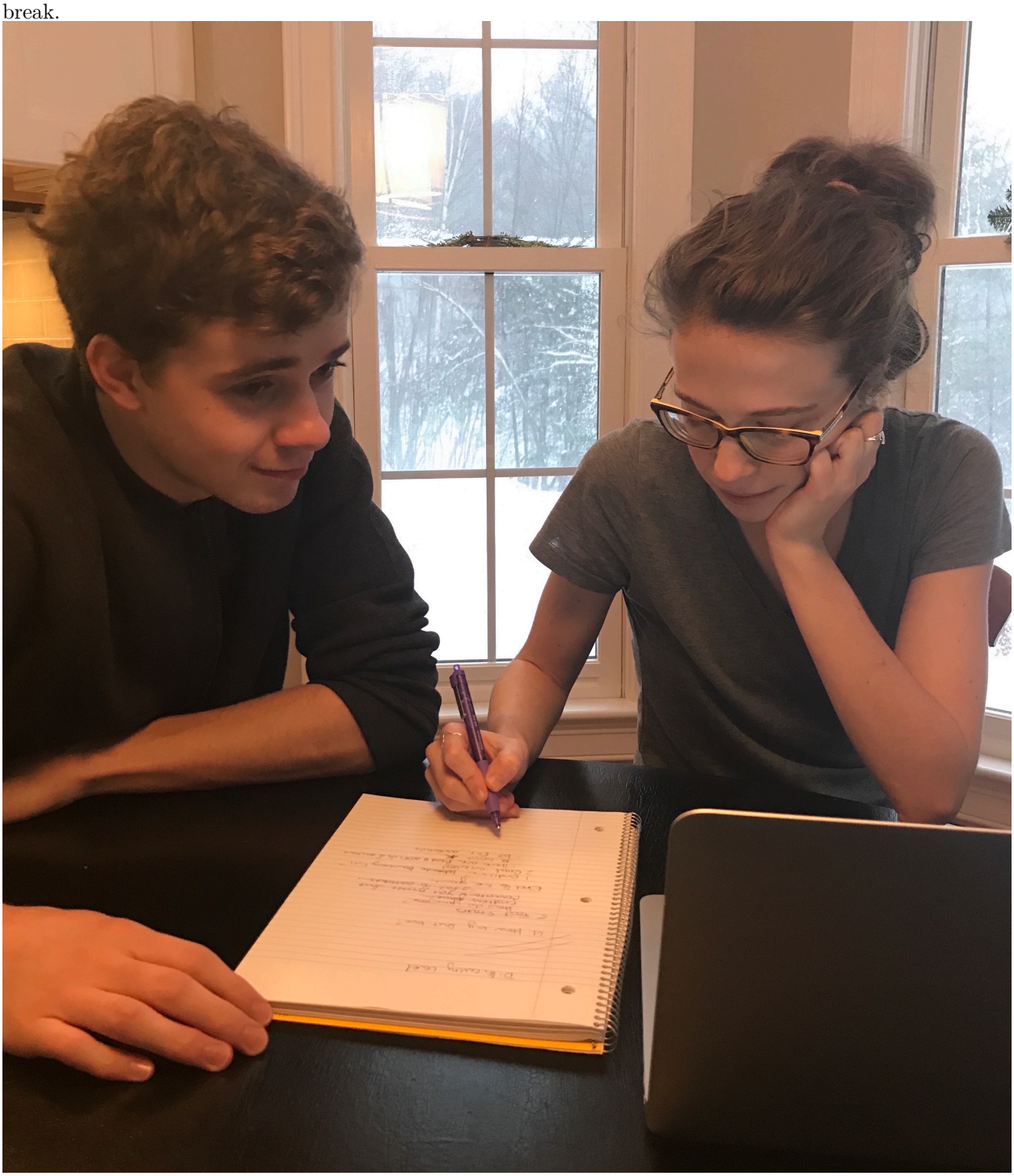
Making a game using codesters

SKH 1/4/2018

Introduction

My brother and I went through the process of creating a game using codesters during a snow day over break.

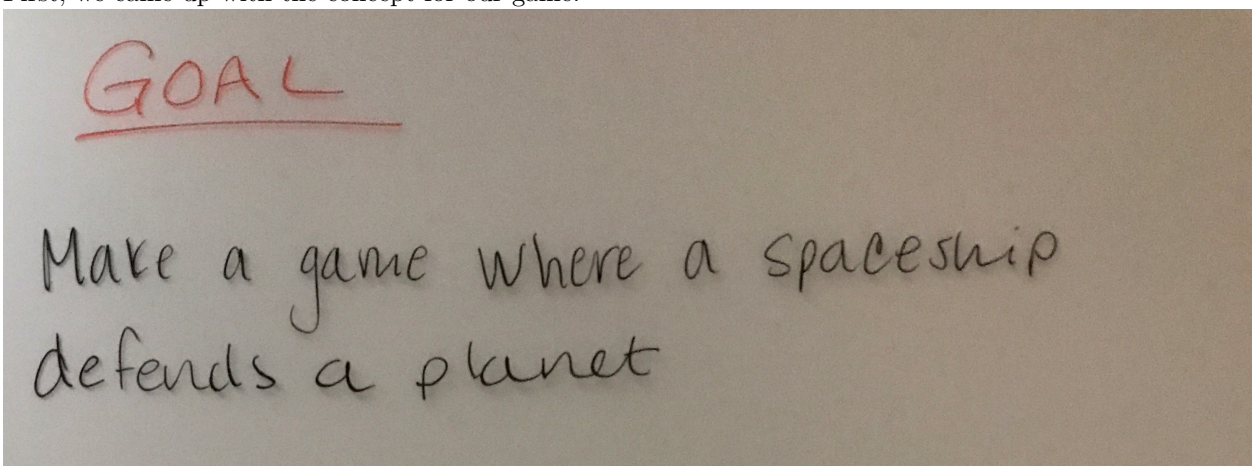


To plan the game, we

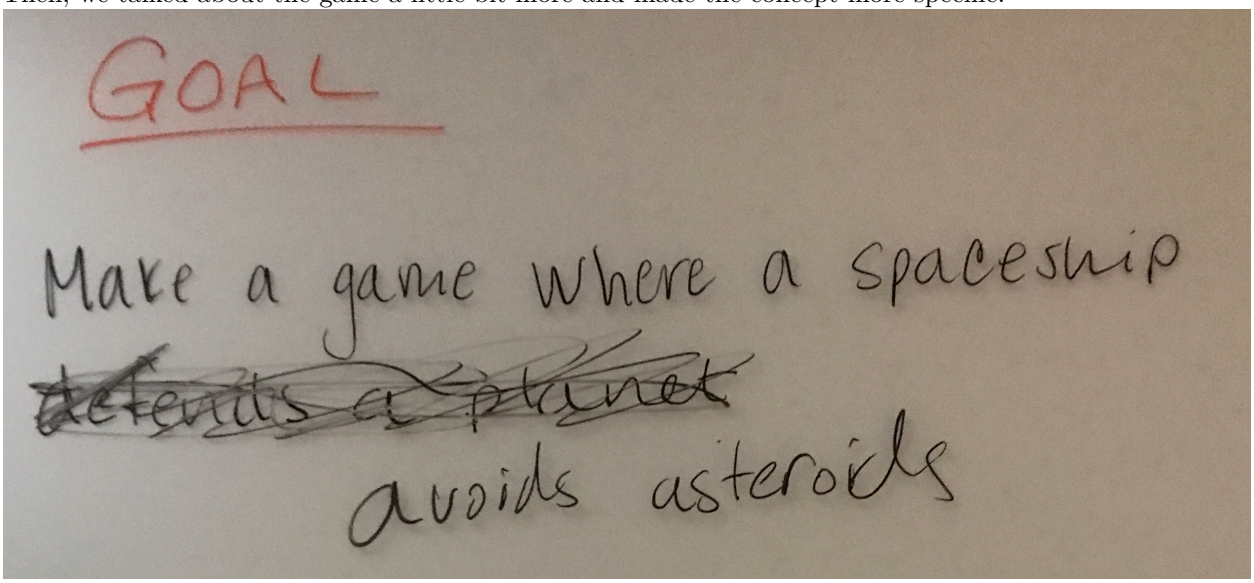
1. Defined a project goal
2. Made a wireframe
3. Made a coarse-grain to-do list
4. Made a fine-grain to-do list
5. Made a "Beyond 'Minimum Viable Product' " list

Define a project goal

First, we came up with the concept for our game:



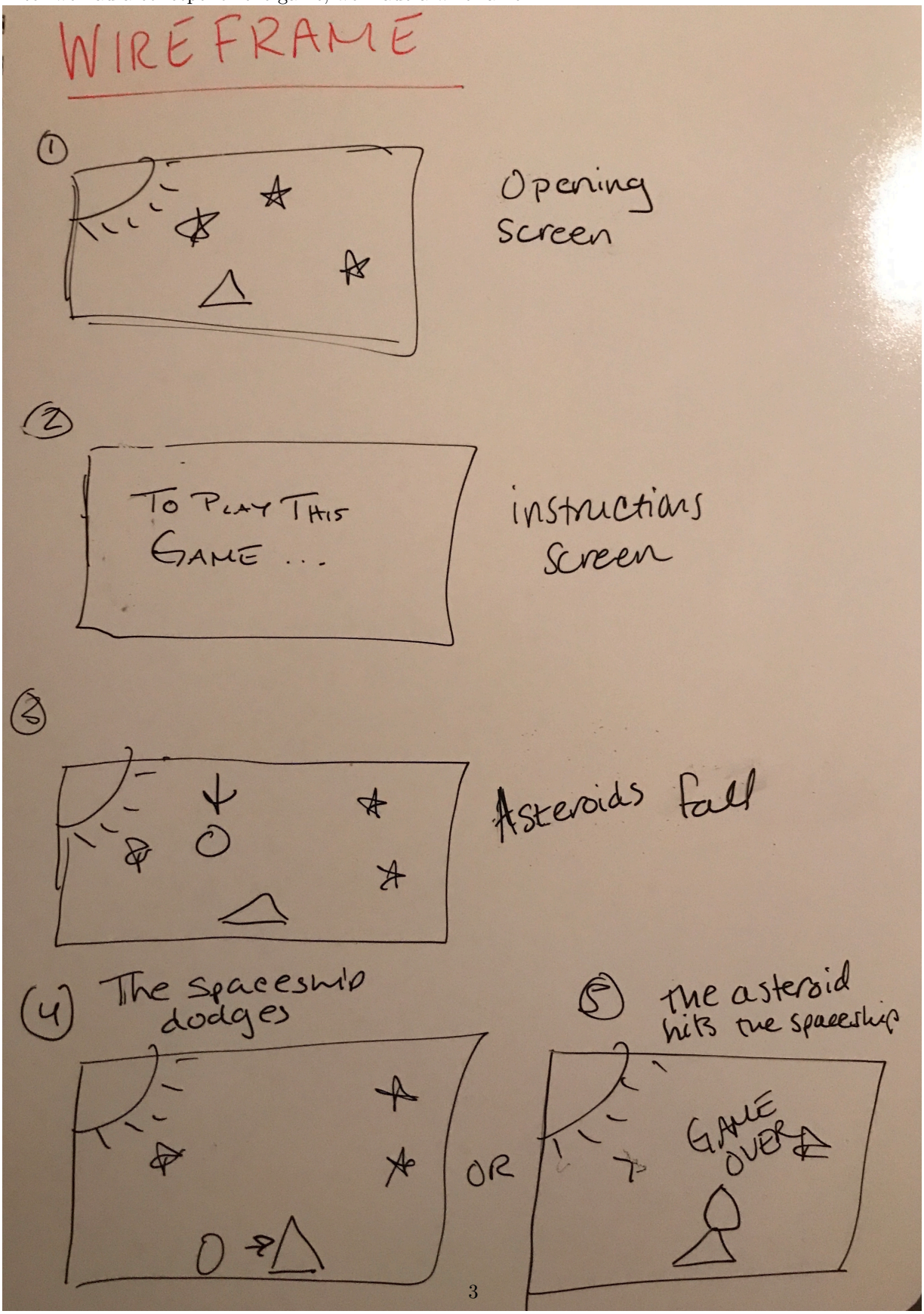
Then, we talked about the game a little bit more and made the concept more specific:



This statement became our "Minimum Viable Product" (MVP).

Wireframe

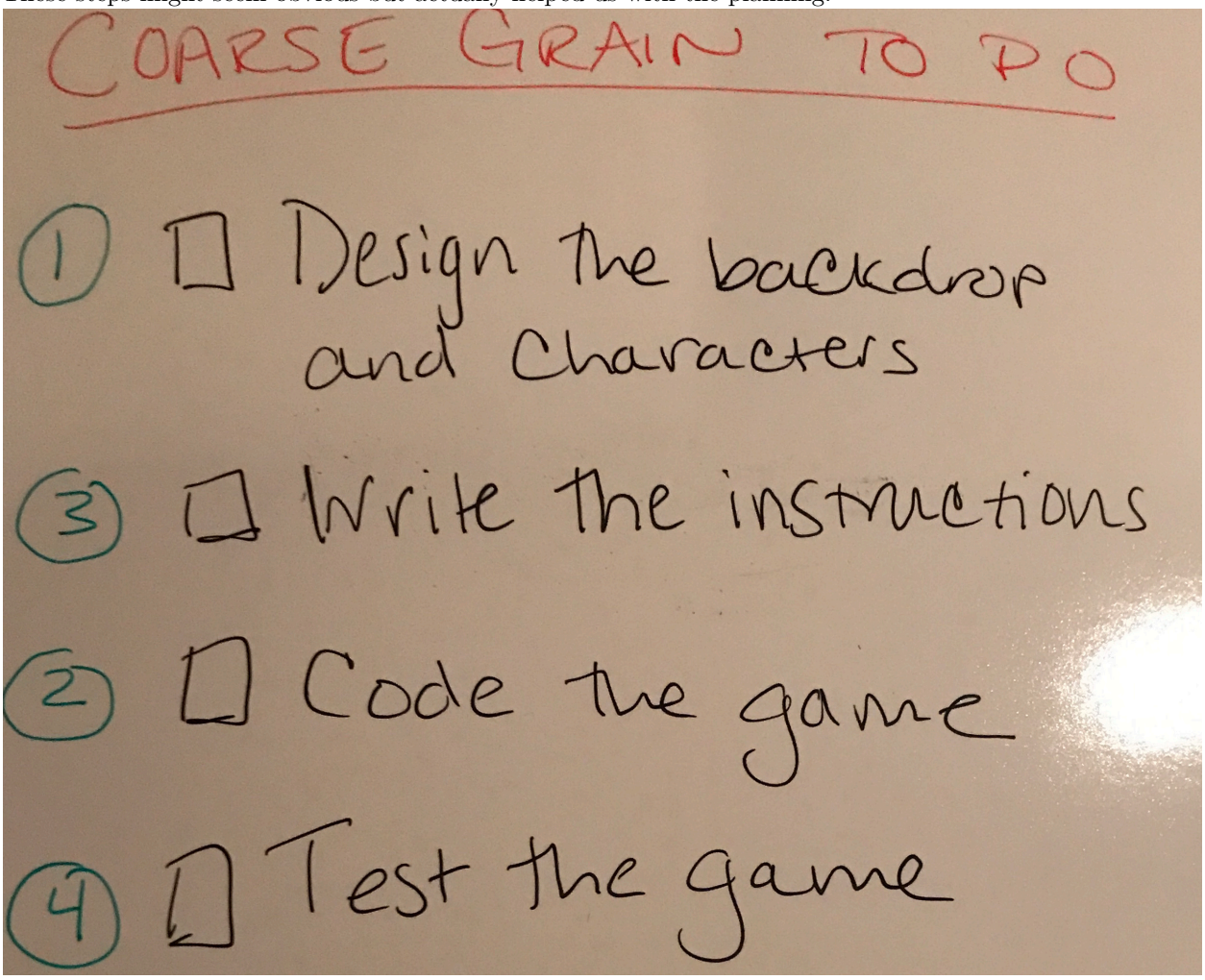
After we had a concept for the game, we made a wireframe:



We showed how we thought the game would start, what would happen if the user did the right thing (dodge the asteroid), and what would happen if the user did the wrong thing (fail to dodge the asteroid).

Coarse-grain to-do list

The next thing we did was make a rough outline of the steps we need to complete in order to make the game. These steps might seem obvious but actually helped us with the planning.



For instance, first we thought we would write the instructions and then make the game. But we realized we should make the game first because we might make changes to our plan while we were coding which would affect the instructions.

Fine-grain to-do list

Next, we took two of the items on the course-grain to-do list and made fine-grain to-do lists. These lists had very specific actions or questions. This helped us break down the project into manageable pieces and decide what we would do first.

FINE GRAIN TODO

Design

- [ ] pick a backdrop
- [ ] pick a spaceship
- [ ] pick an asteroid

Code

- [ ] How will the Spaceship move?
  - letter keys?
  - left / right arrow?
  - mouse?
- [ ] How will the asteroids move?
  - Straight down?
  - Same speed?
- [ ] How does the game end?
  - You get hit once?
  - twice? etc?

We looked through the “Sprites” and “Stage” on `codesters` to complete the action-items under “Design”.

We looked under “events”, “physics”, and “actions” on `codesters` and talked about how we wanted the game to go to complete the action-items under “Code”.

FINE GRAIN TODO

Design

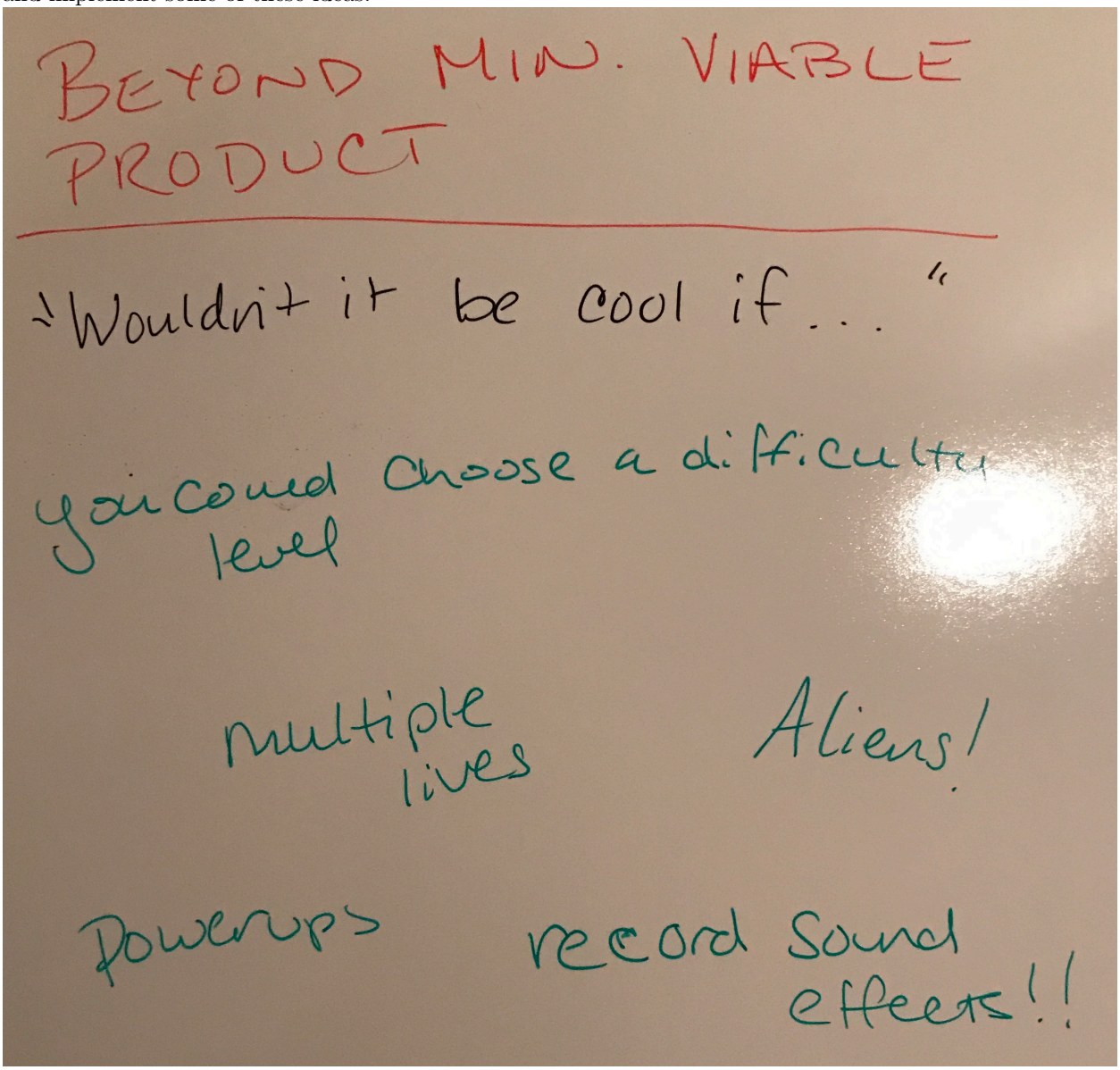
- [ ] pick a backdrop
- [ ] pick a spaceship
- [ ] pick an asteroid

Code

- [ ] How will the Spaceship move?
  letter ~~keys~~?
  left / right arrow?
  mouse?
- [ ] How will the asteroids move?
  Straight down? yes
  Same speed? yes
- [ ] How does the game end?
  You get ~~hit~~ once?
  twice? etc?

Beyond MVP brainstorm

During this process we came up with lot's of different ideas but many of them went beyond our "Minimum Viable Project" (MVP). We put them on a separate list. After we make the basic game we might go back and implement some of these ideas.



Final product

Click HERE to see the final game!

More details on coding in `codesters`

`Codesters` is written in a `python`-like syntax. This means that the code looks a lot like `python` but a) has some functions built-in which `python` does not and b) cannot do *everything* `python` can do.

Below I'll walk through some of the logic behind the different parts of the `codesters` code for my rocket game.

Objects

`python` is a type of programming called **object-oriented programming (OOP)**. If you want to learn more about OOP, this online tutorial is a great resource. For the purposes of making a `codester` game, you don't need to understand the nitty-gritty of OOP, it just helps you do cool things really easily!

In `codesters`, the background (the `stage`) and the characters (the `sprites`) are a special type of data structure (think: container) which contain `variables` **and** `functions`. What this means practically is that you can get information about a sprite and make the sprite do things using the easy syntax `sprite.<SOMETHING>`.

For example (using psuedocode), you might type `print sprite.position` and the output would be the coordinates of the sprite on the screen. This would be an example of the sprite storing a `variable` (the coordinates). You could type `sprite.move_left()` and the sprite would move to the left. This is an example of the sprite storing a `function`.

Below, are some specific examples of `stage` and `sprite` functions and variables in the rocket game.

Stage

In `codesters`, the background is called the `stage`.

You may want to change the background image of the `stage`. You would do that using the line:
`stage.set_background("space")`.
Look around on the `codesters` website, where do you find the list of choices for the background?

You may to have the game pause for a couple seconds, maybe to give the user time to read some instructions you have printed on the screen. You would do that using the line:
`stage.wait(5)`
and the game will pause for 5 seconds. **How would you have the game pause for 10 seconds?**

Sprites

In `codesters` you only have one `stage` but you can have as many `sprites` as you want!

When you call a new `sprite`, you want to give it a *unique* name so that you (and the computer) can tell them apart. For the rocket game, I made a sprite called `rocket` which looks like … a rocket!
`rocket = codesters.Sprite("rocket")`
Look around on the `codesters` website, where do you find the list of choices for the `sprites`?

You can change the position of the `sprite` using the `function` `glide_to`:
`rocket.glide_to(0,-450)`.
What do you think would happen if you changed 0 to 100?

You can retrieve the coordinates of the `sprite`:
`current_x = rocket.get_x()`.
If you called `rocket.glide_to(0,-450)` and then `current_x = rocket.get_x()`, what would be the value of the variable `current_x`?

The sprites can talk!
`rocket.say("ow")`

Events

Knowing what we know so far, we can have the sprites move around the screen but we don't know how to let the user interact with the sprites. For instance, what if we don't just want the sprite to

`rocket.glide_to(0,-450)` but we want it to glide to the X-Y coordinates the user specifies? Or move based on a key the user hits? To do this, we have to use "listeners".

Listeners

When you set up a "listener", the computer waits for a specific user input (ie "left key") and when it "hears" that input calls another function.

For example, this line of code
`stage.event_key("left", move_spaceship_left)`
says "when the user hits the 'left' key call the function `move_spaceship_left`". I'll go through the contents of `move_spaceship_left` below. **What would happen if this line was changed to stage.event_key("space", move_spaceship_left)?**

Sometimes, the "listeners" listen for events other than user inputs. For example, what happens if two sprites hit each other?
`rocket.event_collision(collision)` This line of code says "when the `sprite` called 'rocket' hits another `sprite`, call the function `collision`". I'll go through the contents of `collision` below.

Functions

In `codesters`, you can write your own functions, just like any other language.

One function I wrote for the rocket game was a function which dictates what happens when the user hits the left arrow.

```
def move_spaceship_left():
    # figure out *where* it is
    current_x = rocket.get_x()
    y = rocket.get_y()
    if current_x > -500: #edge of screen
        rocket.glide_to(current_x - rocket_speed, y)
    else:
        rocket.glide_to(current_x, y)
```

Why did I write `current_x - rocket_speed` instead of `current_x + rocket_speed`?

I also wrote a function to dictate what happens when another `sprite` hits the rocket.

```
def collision(asteroid, hit_sprite):
    rocket.say("ow")
    text = codesters.Text("Game Over!", 0, 100, "yellow")
```

What would I do if I wanted the asteroid to say 'ow'?

Other examples

Another game

Click HERE (I didn't write this one)

Trivia

Click HERE