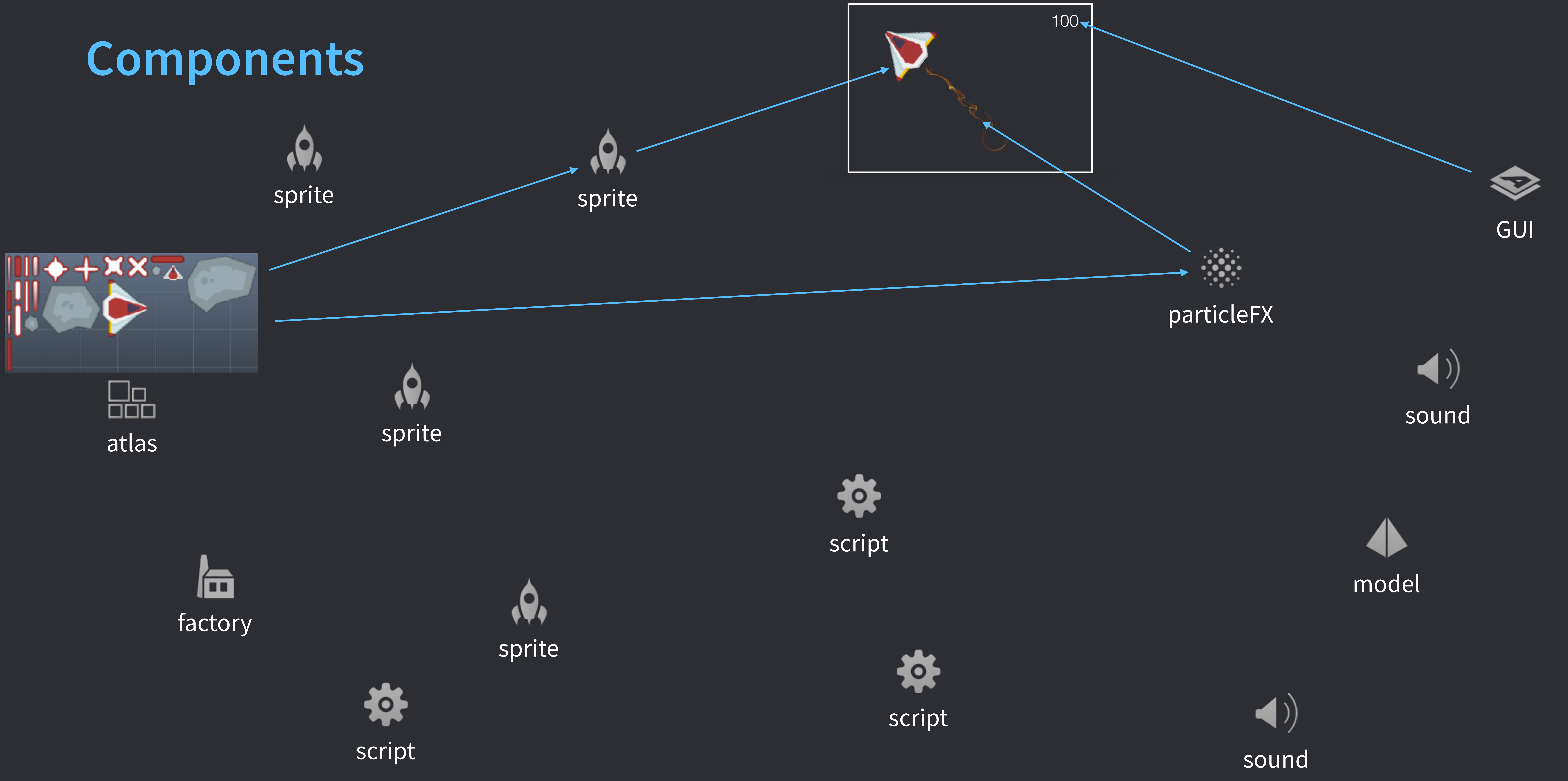
Core concepts

London, January 2017

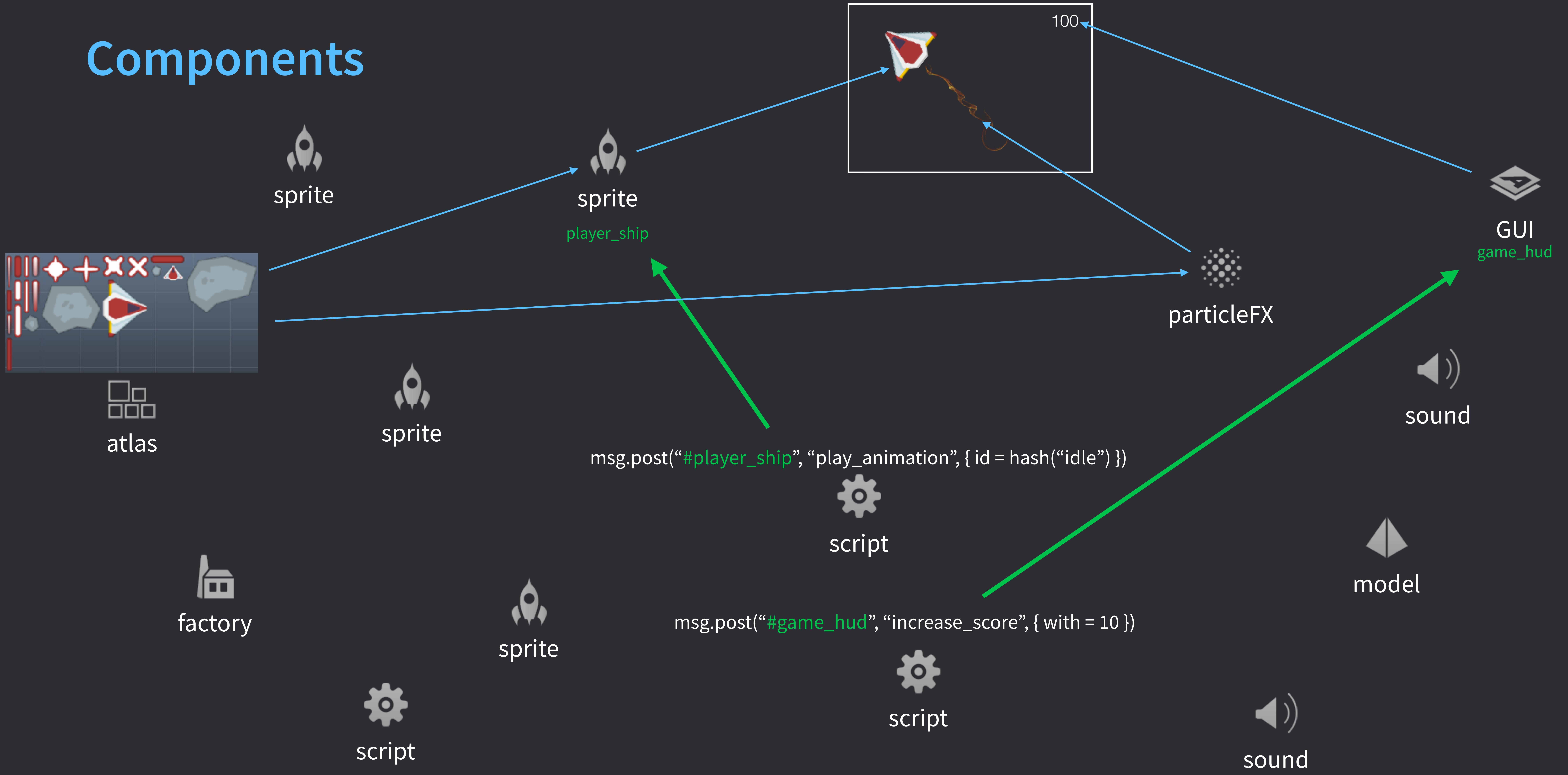
What are we doing tonight?

- Creating a project
- Components, game objects and collections
- Message passing and URL:s
- Parents, children and the scene graph
- Lifecycle
- Factories
- Data, scopes and “self”
- Object orientation
- Hot reload

Components


100
sprite
sprite
GUI
particleFX
atlas
sprite
sound
script
model
factory
sprite
script
script
sound

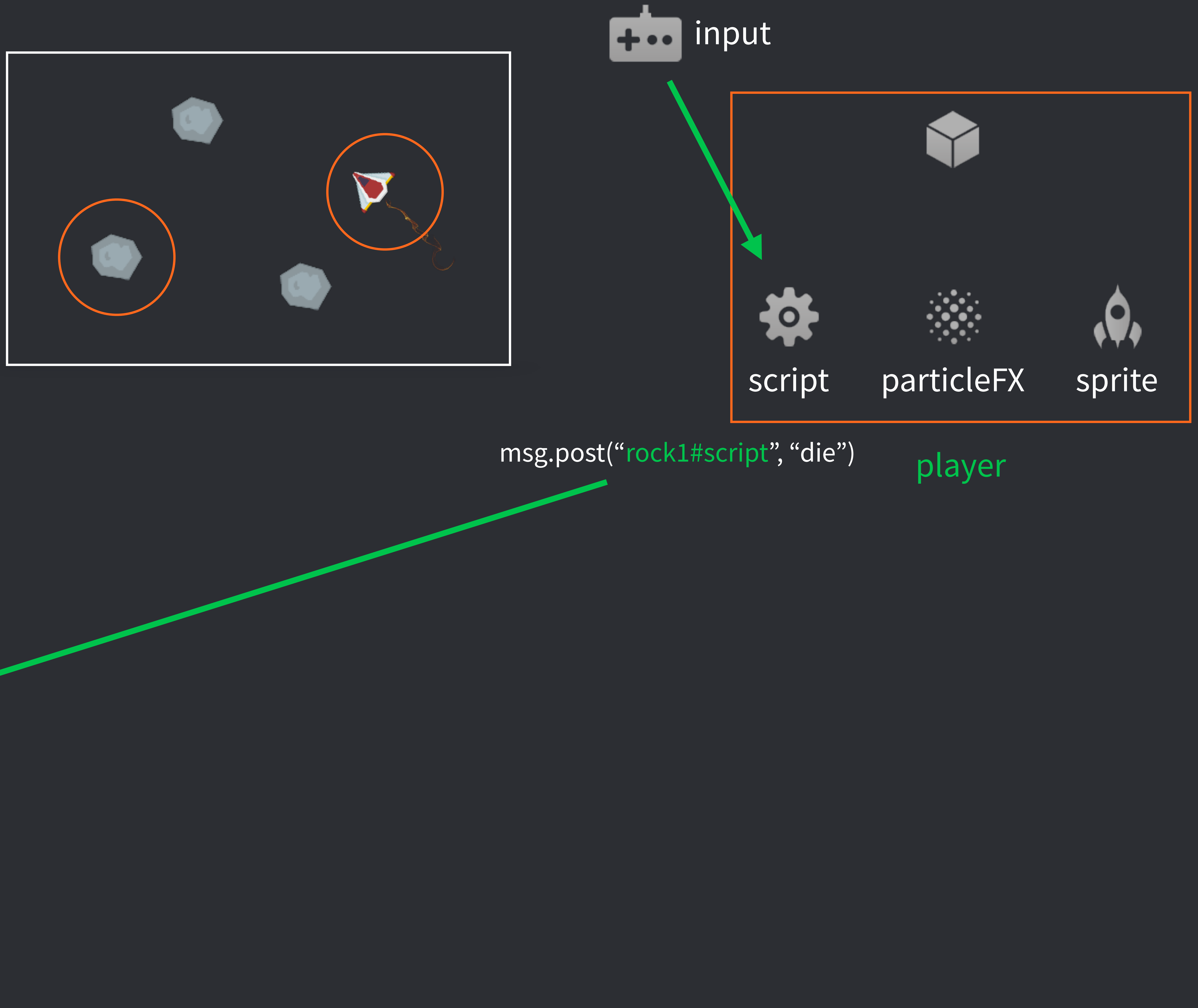
Components



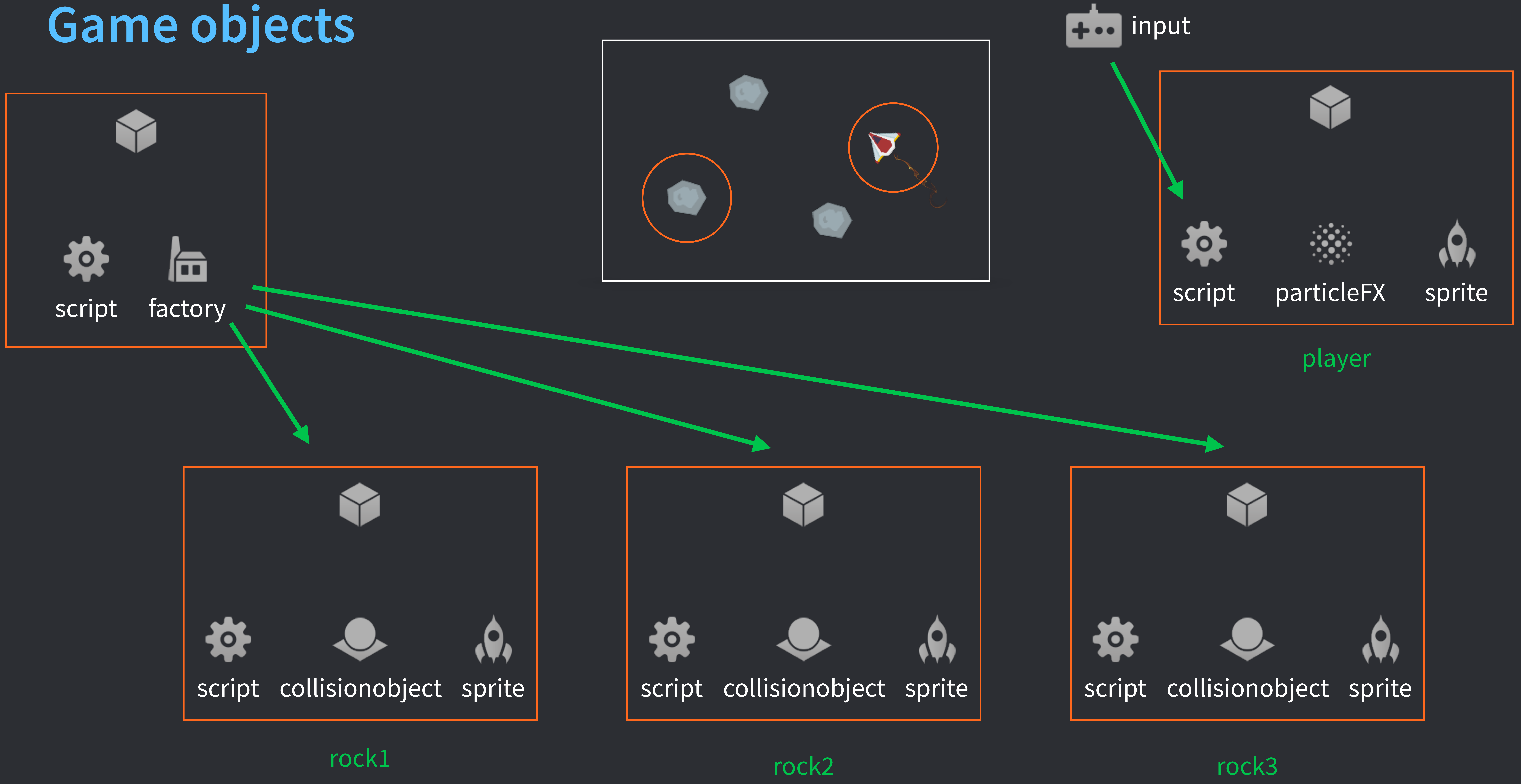
```
msg.post("#player_ship", "play_animation", { id = hash("idle") })
```

```
msg.post("#game_hud", "increase_score", { with = 10 })
```

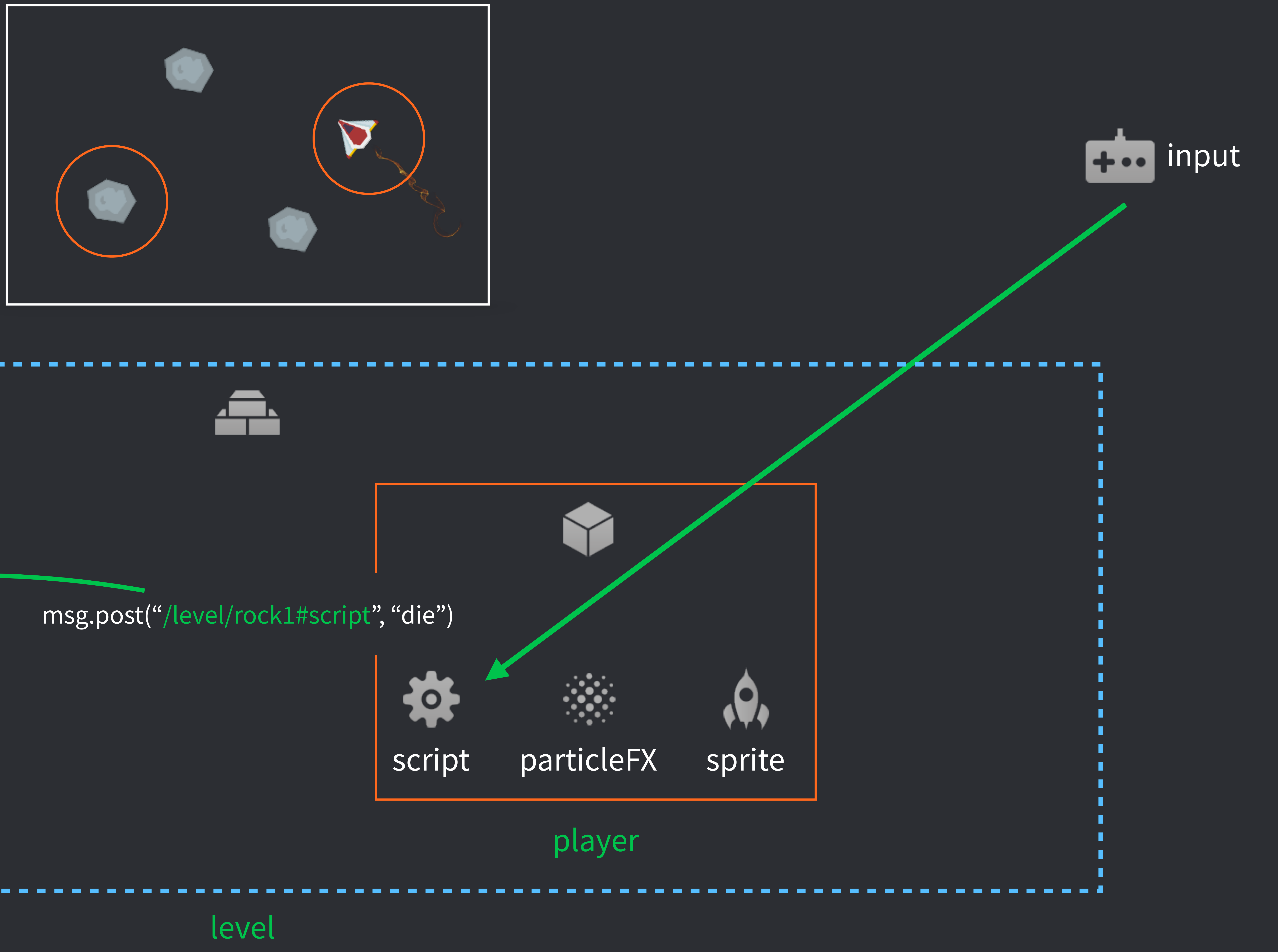
Game objects



Game objects

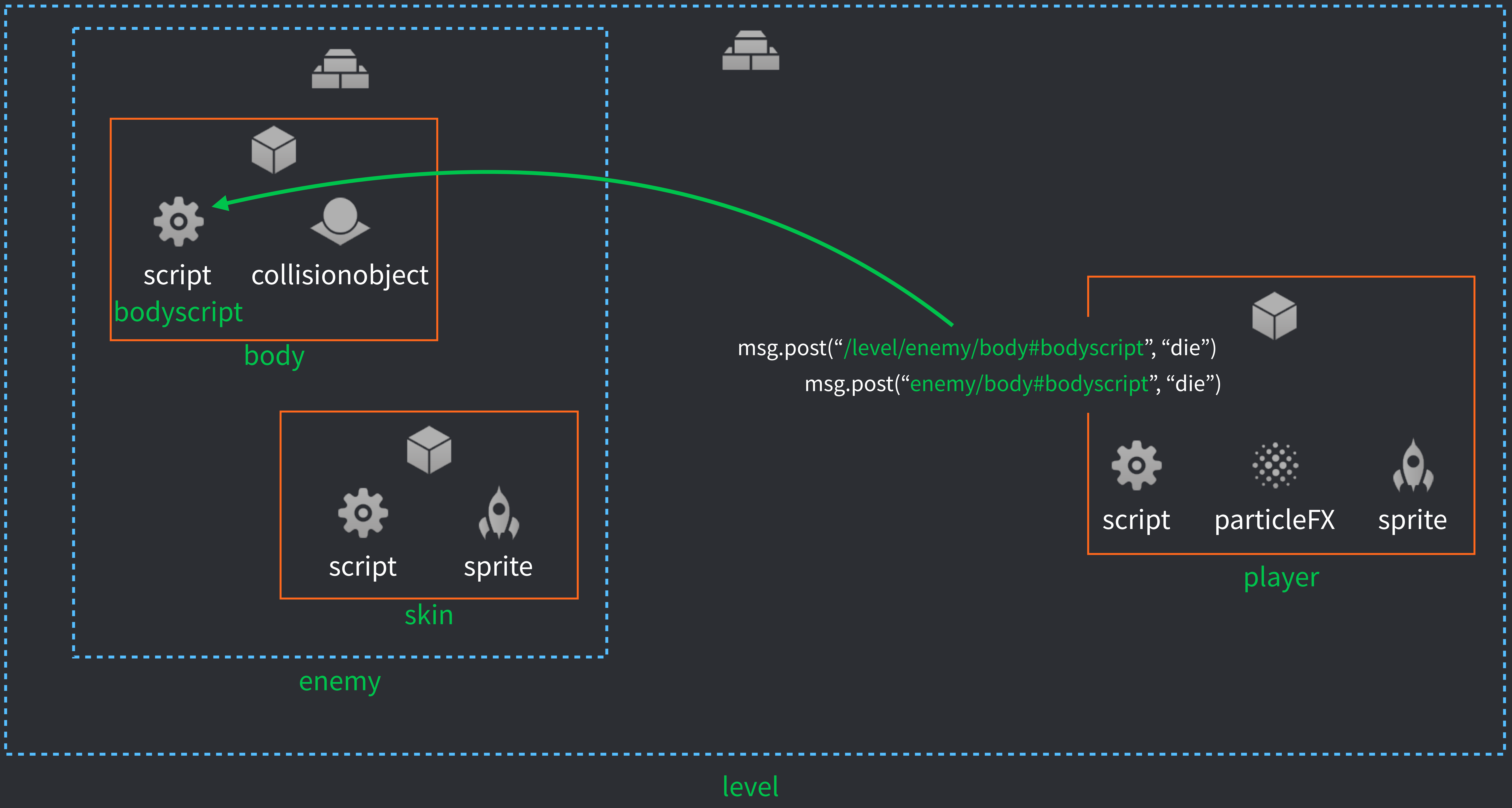


Collections



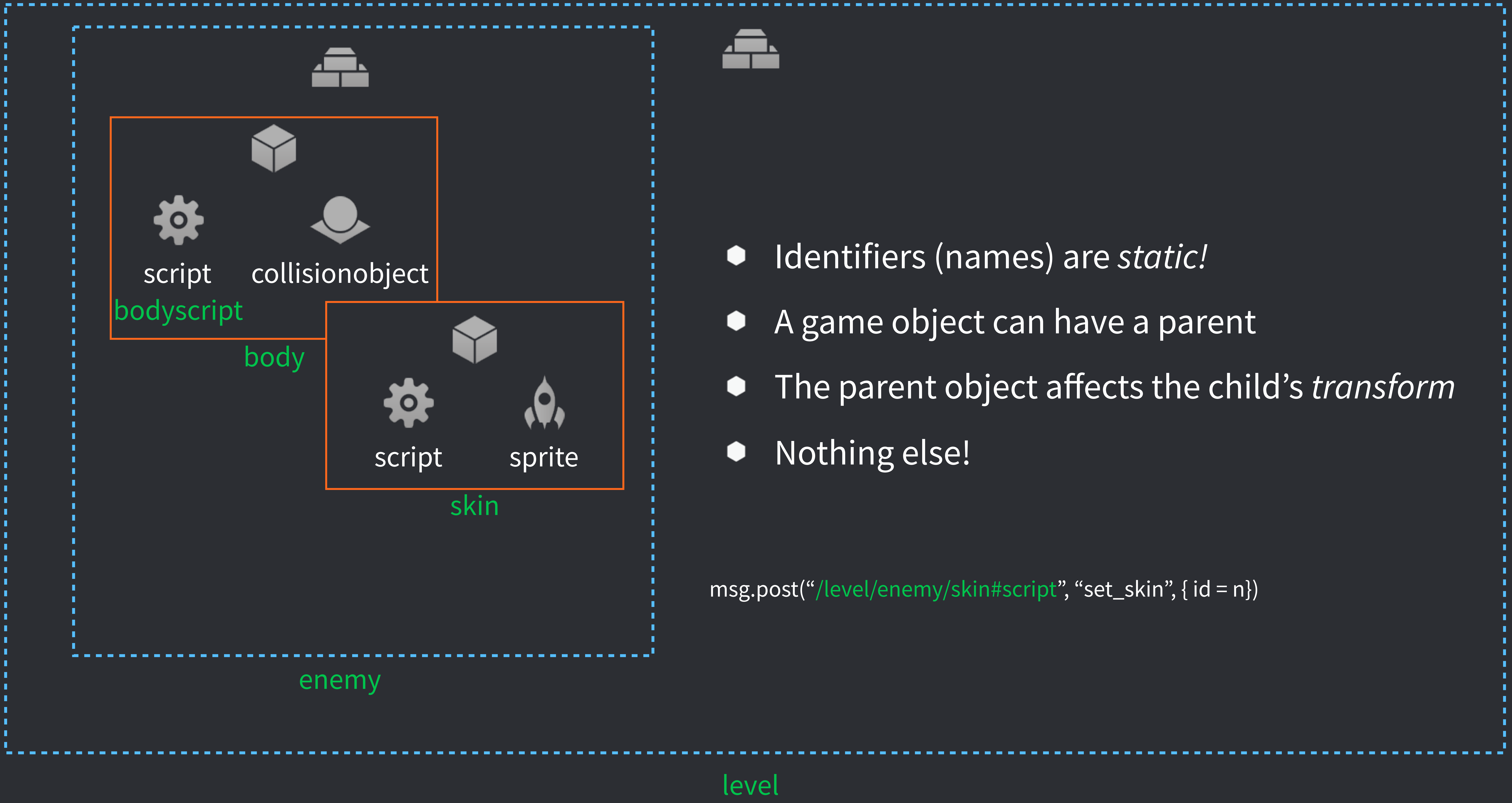
msg.post("/level/rock1#script", "die")

Collections



```
msg.post("/level/enemy/body#bodyscript", "die")
msg.post("enemy/body#bodyscript", "die")
```

Parent-child relations



- Identifiers (names) are *static!*
- A game object can have a parent
- The parent object affects the child's *transform*
- Nothing else!

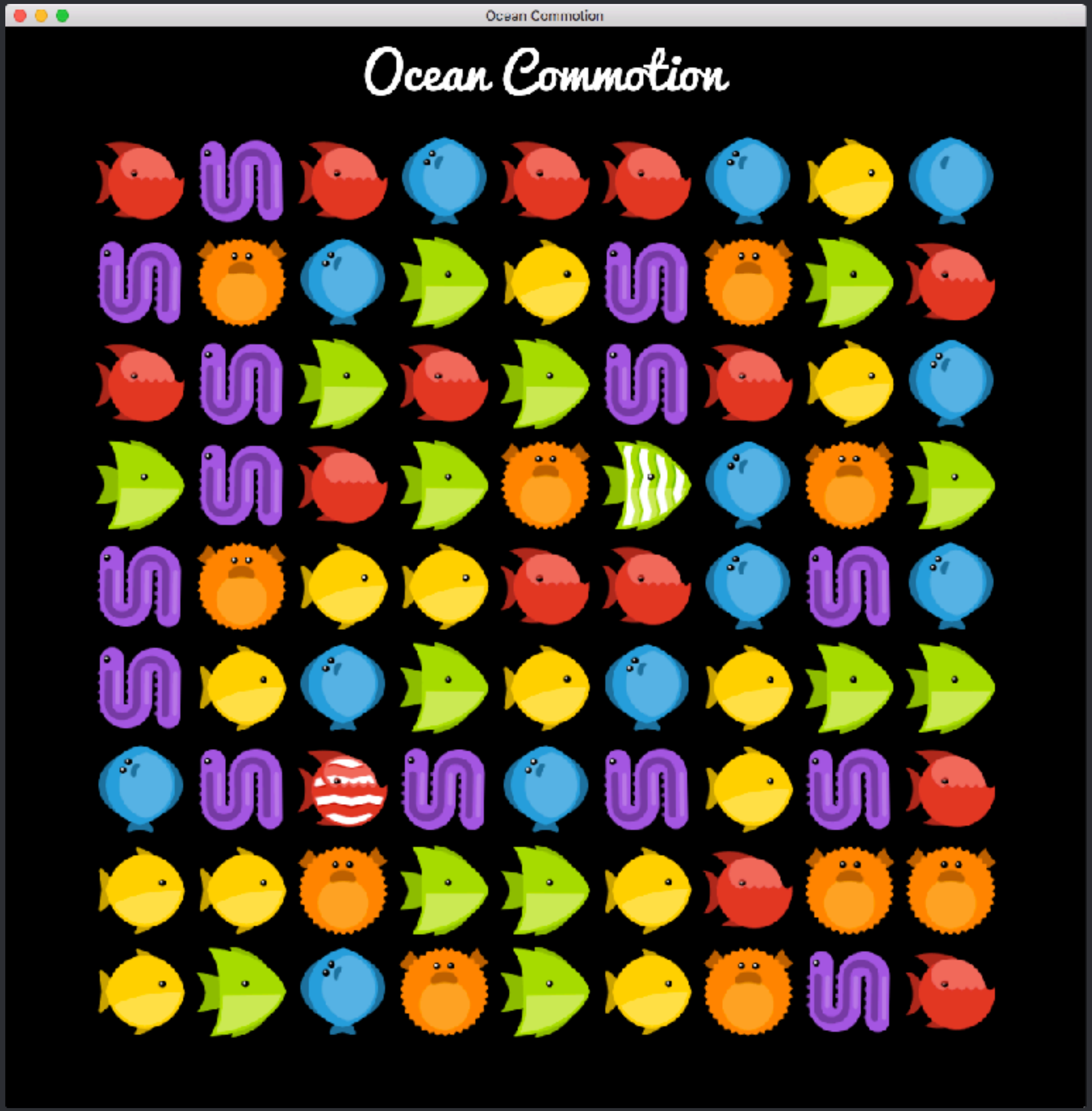
```
msg.post("/level/enemy/skin#script", "set_skin", { id = n})
```

Lifecycle

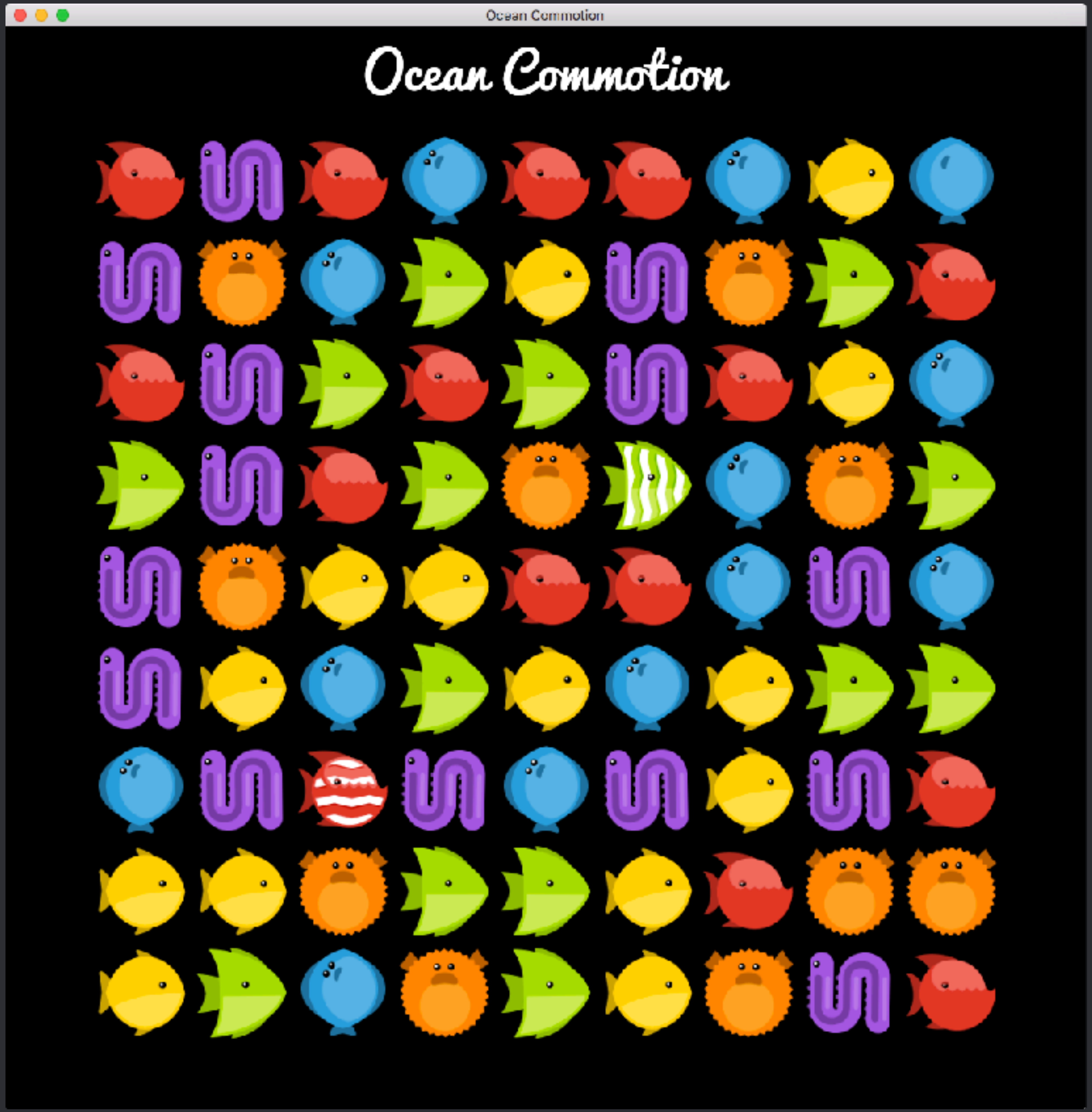
Factories

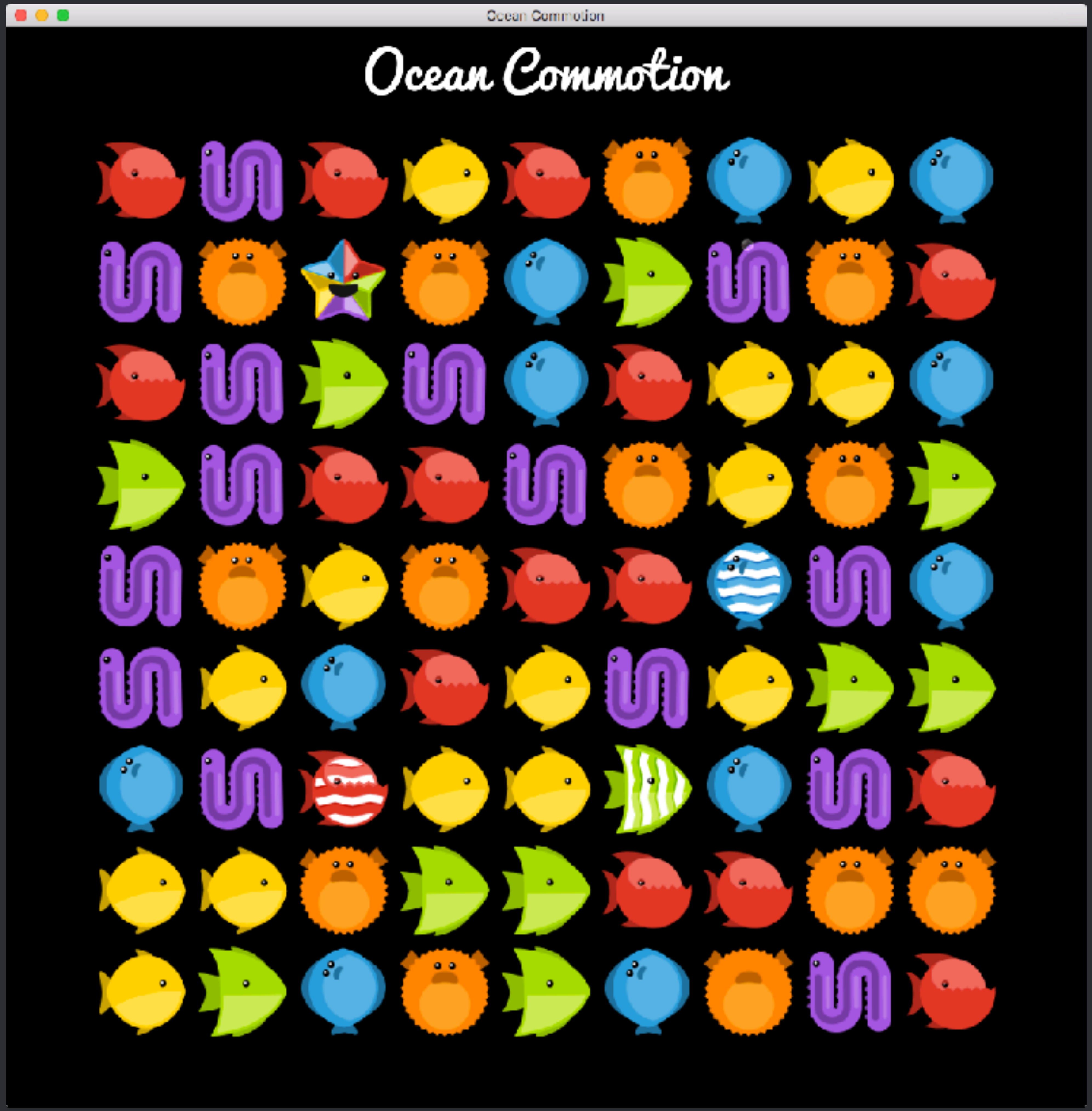
Data, scopes and “self”

Object orientation

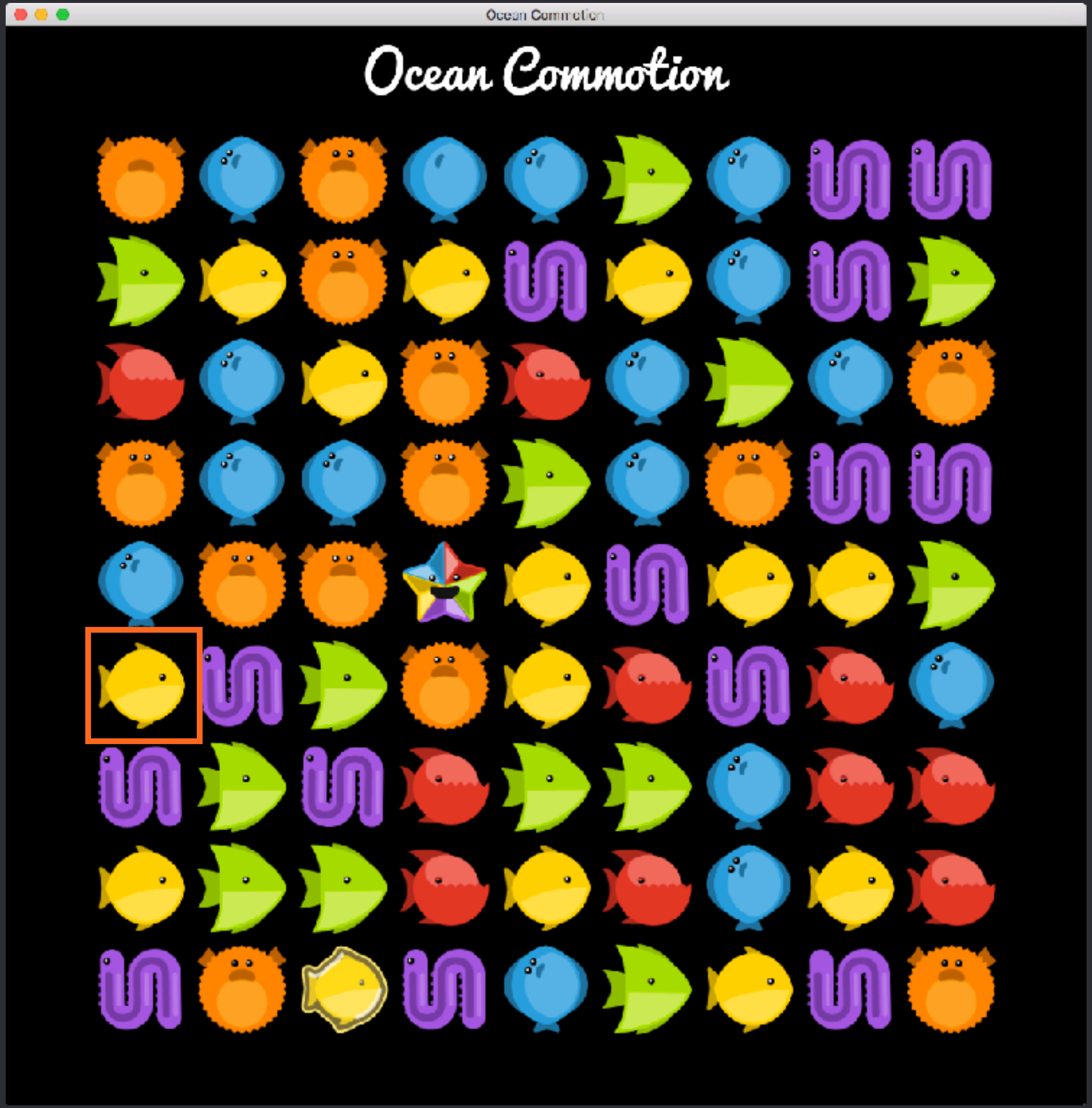


A game: transformation of *data*





Data representing the game

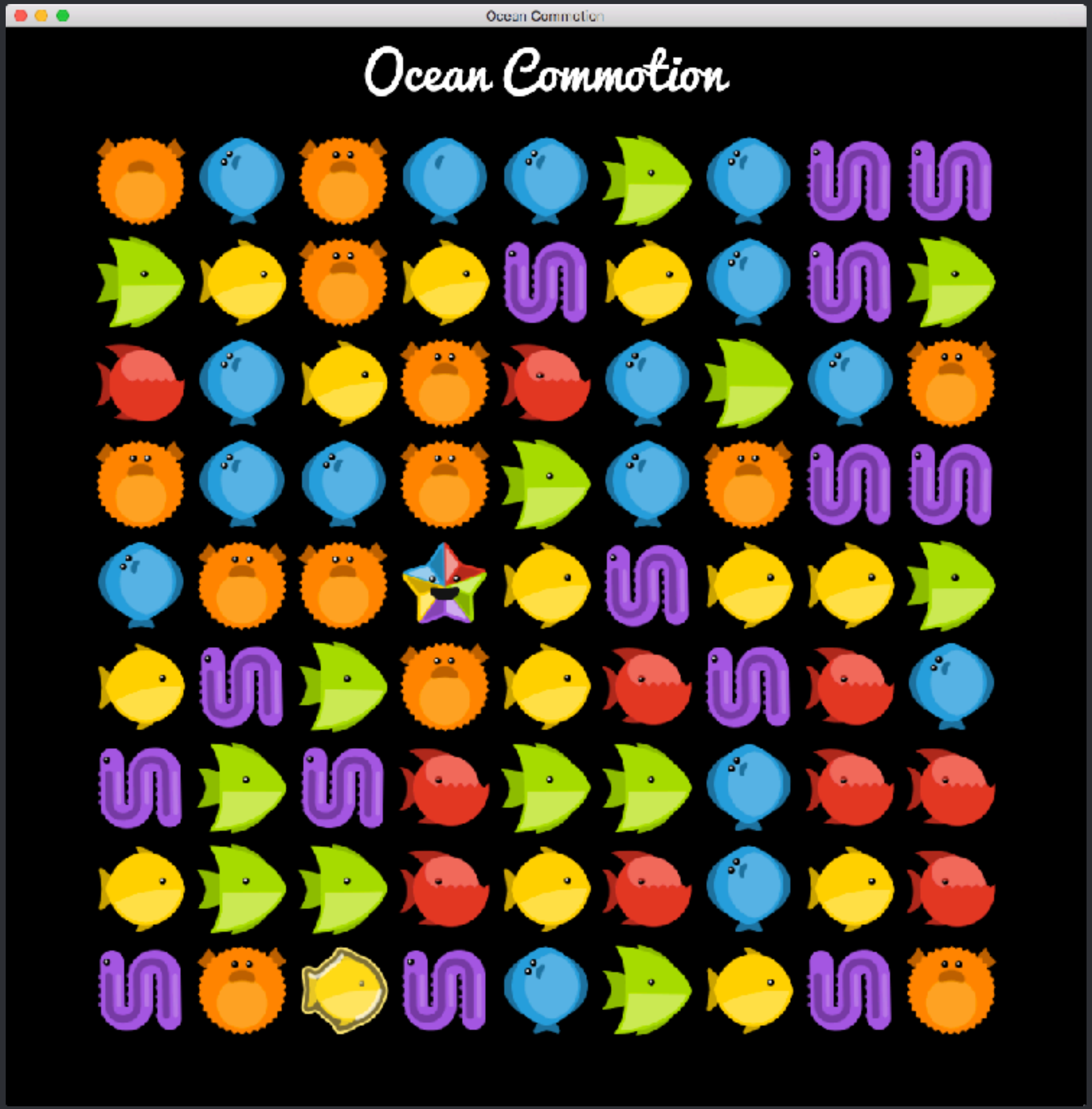


```
0 = {
   0 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [purple],
     y = 0,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance1],
   }
   1 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [yellow],
     y = 1,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance2],
   }
   2 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [purple],
     y = 2,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance3],
   }
   3 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [yellow],
     y = 3,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance4],
   }
   4 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [blue],
     y = 4,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance5],
   }
   5 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [orange],
     y = 5,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance6],
   }
   6 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [red],
     y = 6,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance7],
   }
   7 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [green],
     y = 7,
     neighbors_horisontal = {
     }
     type = hash: [plain],
     id = hash: [/instance8],
   }
   8 = {
     x = 0,
     neighbors_vertical = {
     }
     color = hash: [orange],
     type = hash: [plain],
     neighbors_horisontal = {
     }
     y = 8,
     id = hash: [/instance71],
   }
}
```

Game objects in Defold

- ***No classes or inheritance!***
- All objects have their own internal state, memory that they control (through the ***self*** reference)
- Message passing is used to communicate between objects
- You can send ***any*** message to ***any*** existing object
- It is up to your code how to respond to the message (in the `on_message()` function)
- If your code does not contain code for a particular message, nothing happens

Separating concerns



Game object:

- visual representation (sprite)
- does not need to know anything about the game!

High level messages

Game logic are operations on the board data structure

General tips

- Separate concerns: data and logic
- Don’t pass data around that belongs in one place
- Build game objects that does not require context
- Use message passing for high level logic

Hot reload

Thank’s for today!