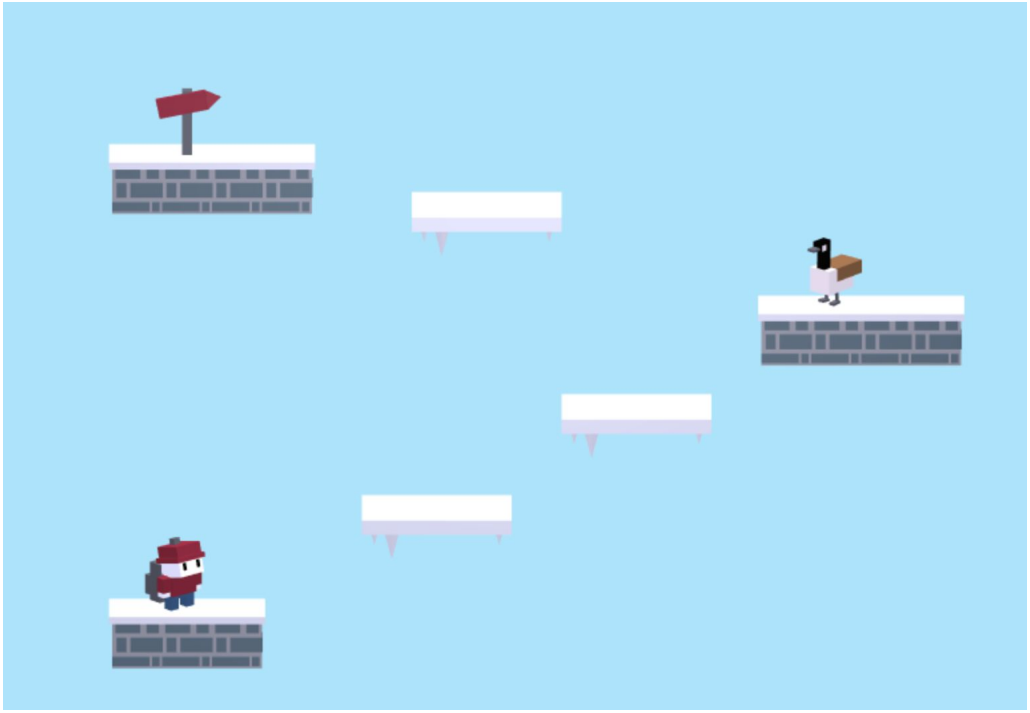
Climbing the Hill



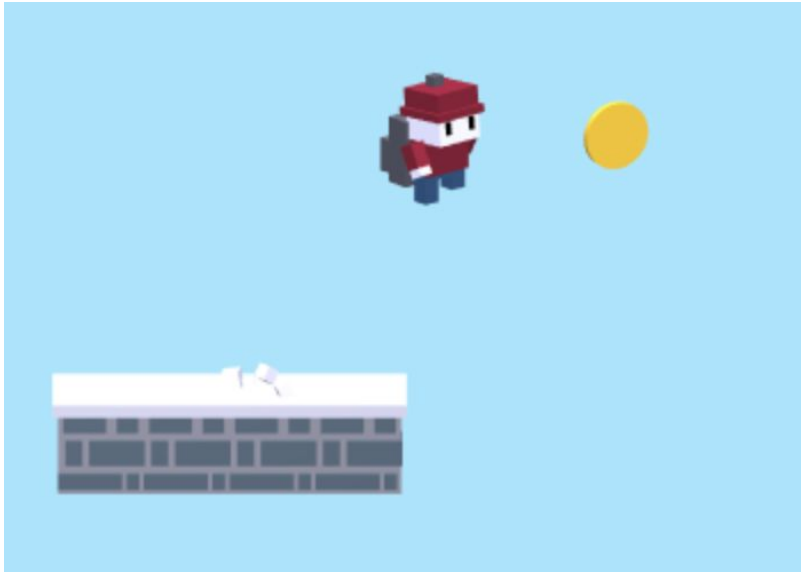
Alex Rosenthal, Bryan Vaihinger, Tino Zinyama

Premise

- A platformer inspired by climbing the Colgate hill
- 2D game logic, graphics made of simple 3D shapes
- Adapt elements of colgate life into game mechanics
- Vertical levels, goal is to reach the very top

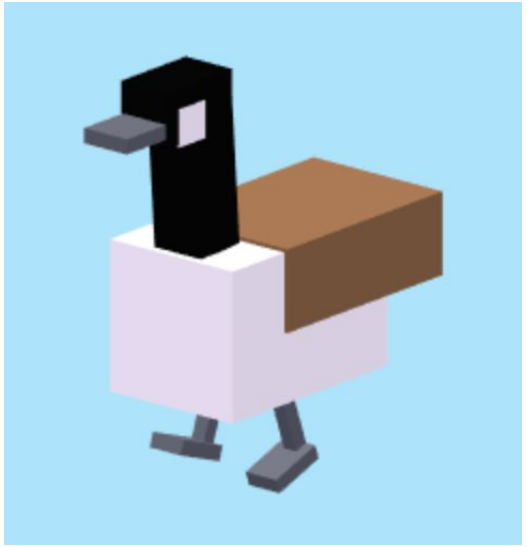
Developing Movement

- Satisfying movement is vital
- Precise maneuverability, but not too jarring
- Jumping is complicated
- The double jump adds a new dimension to gameplay



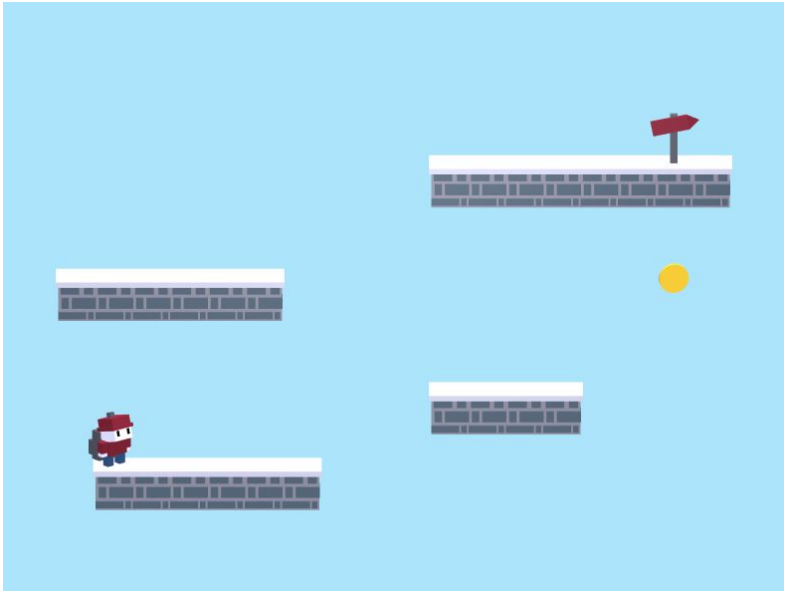
Modelling

- Models are composed of simple geometric shapes
- Each model is divided into parts that can be individually moved
- Character designs communicate ideas, minimum needed to suggest features
- Colors should feel Colgate-esque, and stand out from background



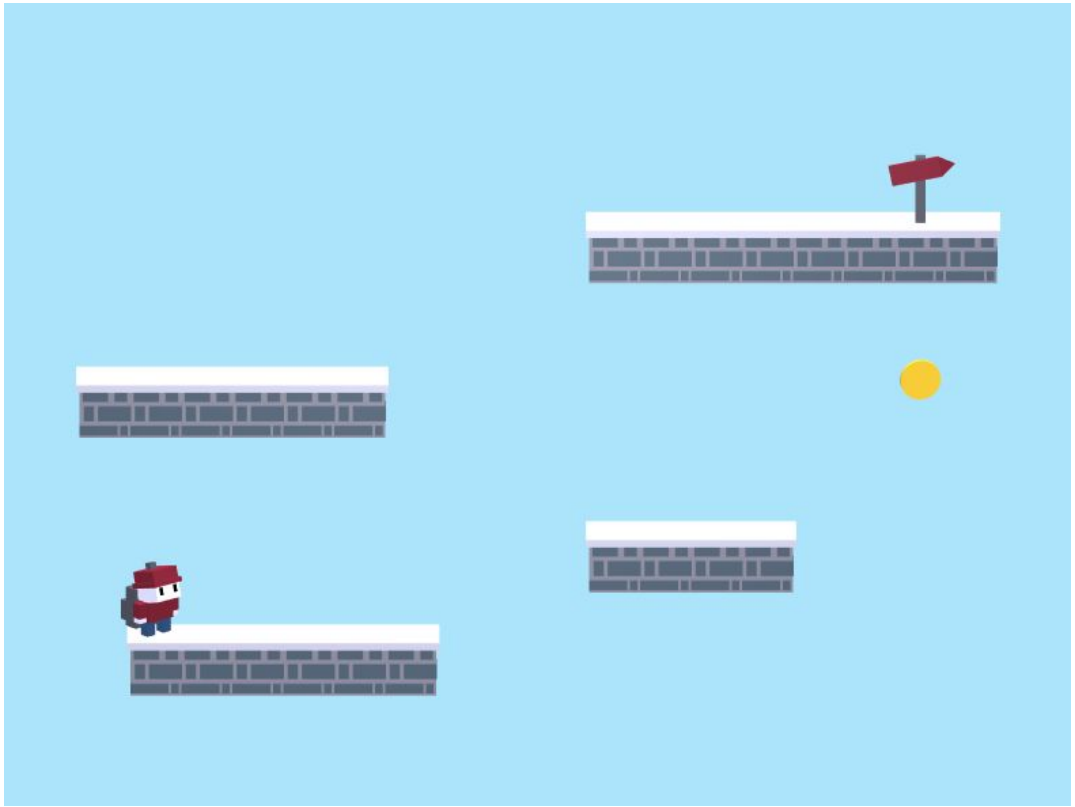
Game Levels

- Adding levels should be easy and should require minimal coding
- So we implemented a text based level editing system
- Each level is defined by a layout string
- Each character in the string maps to a game element



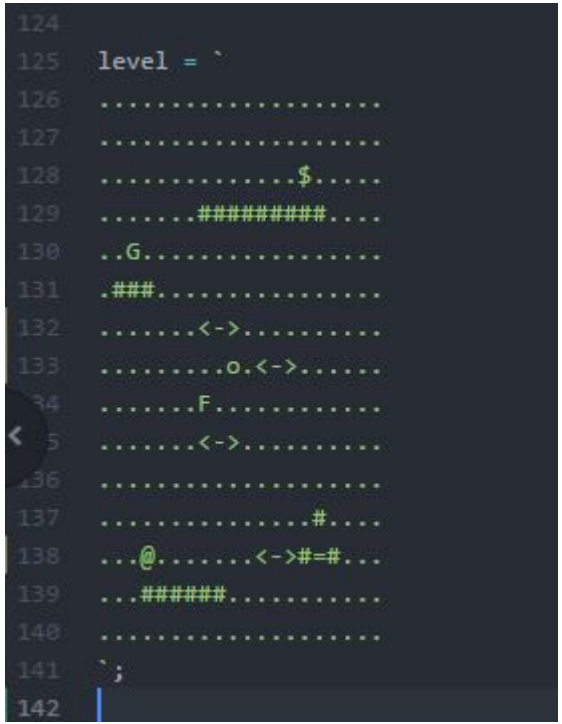
Sample Level

```
47
48  level = `
49  ......................
50  ......................
51  ......................
52  ..................$..
53  ...........########.
54  ......................
55  ......................
56  .######..........o..
57  ......................
58  ......................
59  ...........####......
60  ..@...................
61  ..######..............
62  ......................
63  ......................
64  `;
65
```



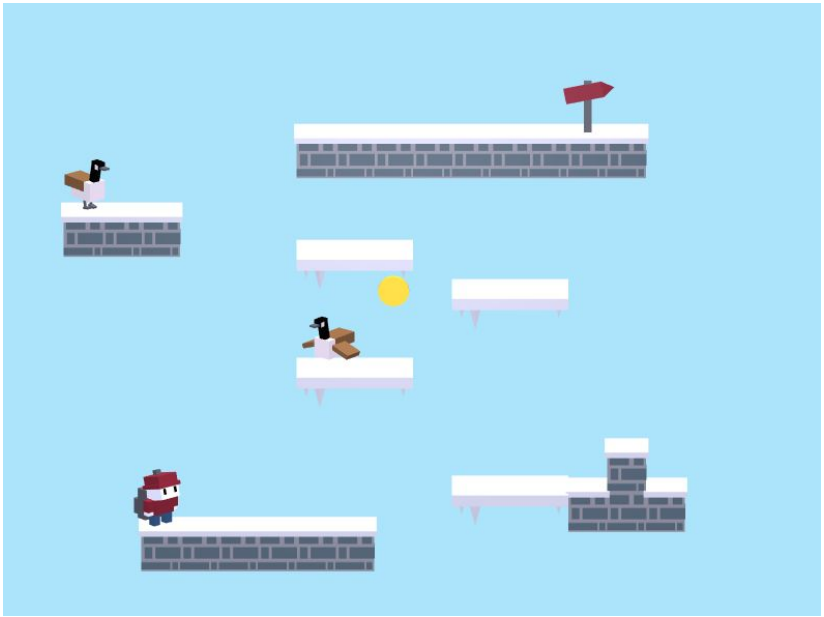
Level Editing

- Level editor reads the layout string and creates objects in the scene accordingly
- Placement of game elements is grid-based
- However, elements can have behaviours that are not tied to the grid



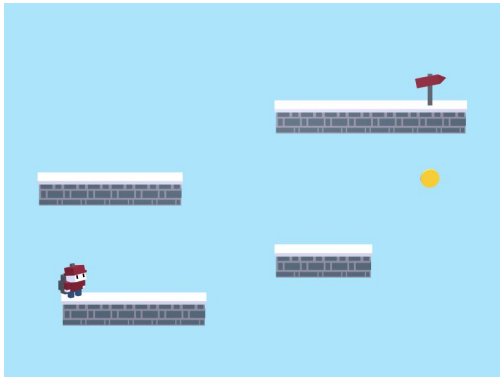
Level Editing

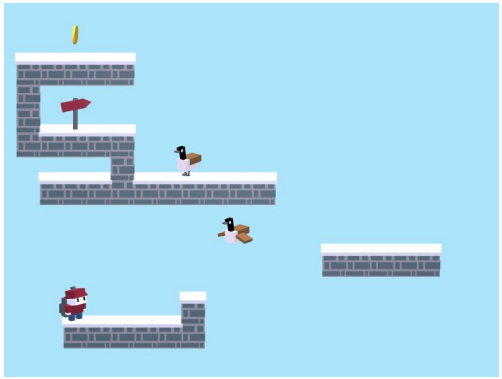
- All game elements created by the level editor implement the same interface
  - init()
  - update()
  - onCollide()
- The game doesn't need to know much about each game object
- Each object specifies its own behaviour

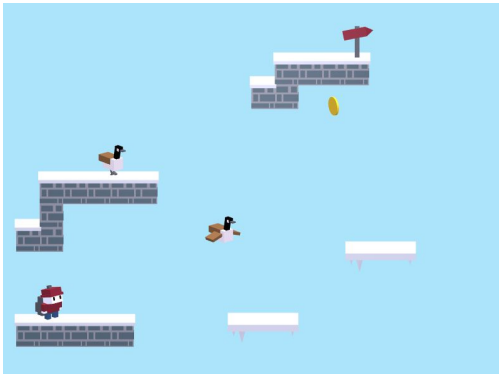


Level Progression

- Levels get harder as the game progresses
- New enemies and obstacles are introduced gradually

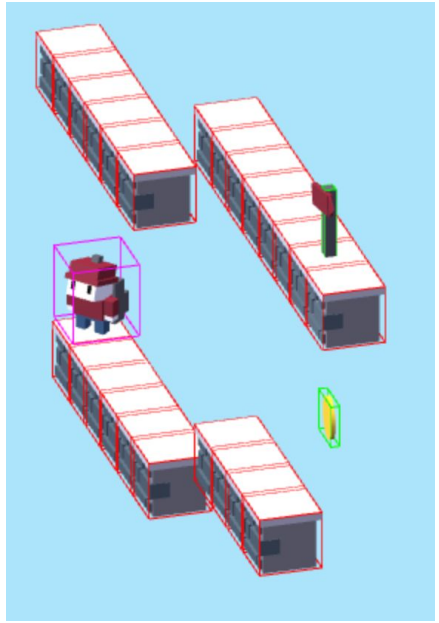






Collisions

- Raycasting vs. bounding boxes
- Physics only occur in two dimensions
- Objects could slip between casts
- Falling through at high speeds



Collisions - Raycasting

```
var checkDown = useMin(checkCol(hBoundingBox[2], dirVectors[2], 0, 2),
    checkCol(hBoundingBox[3], dirVectors[2], 0, 2));

if (checkDown) {
  land();
  this.y += 2-checkDown;
}
  function checkCol(pos, dir, near, far) {
    var ray = new THREE.Raycaster(pos, dir.normalize(), near, far);
    var collisionResults = ray.intersectObjects( collidableMeshList );
    if (collisionResults.length > 0) {
      return collisionResults[0].distance;
    }
  }
```

Collisions - Bounding Boxes

```
113     heroBox.setFromObject(this.model);
114
115     for(var i = 0; i < levelElements.length; i++){
6         checkBox.setFromObject(levelElements[i].model);
7         if (heroBox.intersectsBox(checkBox)){
118         levelElements[i].onCollide();
119       }
120     }
```

```
this.bBoxH = new THREE.BoxHelper(this.collidableMesh, 0x00ff00);
```