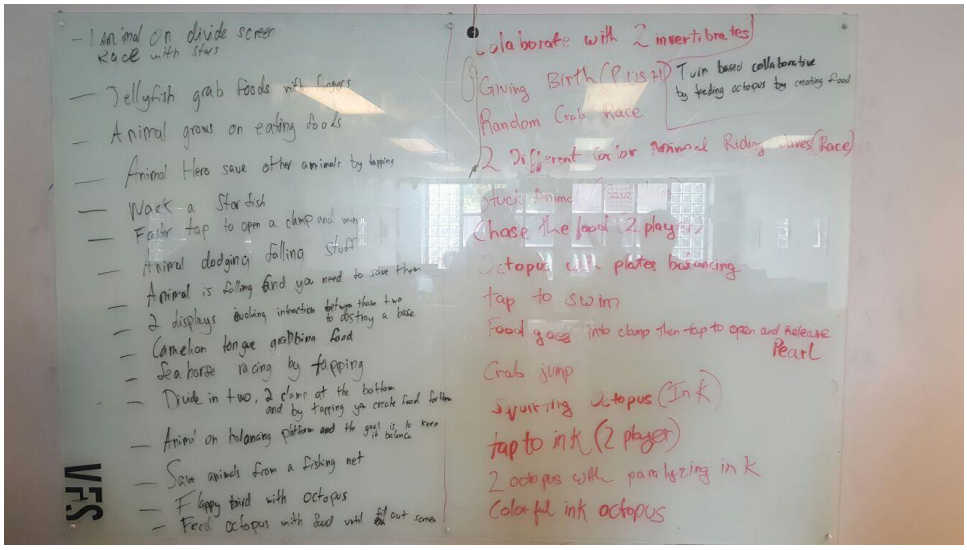
Vancouver Aquarium

Invertebrate exhibit displays

Design Thinking Process

- Empathize: For this step we talked to Nick to get him to open up and relax before we got into the project specifications, we learned more about him:
    - His job: Vancouver Aquarium Exhibit Manager
    - His family: Has a small daughter that loves visiting the aquarium
    - His favorite invertebrate animal in the aquarium: The octopus
- Define: In this stage we talked to Nick about the requirements for the project, we asked as many questions as possible in order to build a base and grasp a direction for the next step. Some of the thing we learned:
    - The project requires a series of displays to entertain the visitors while they wait to see an exhibit.
    - The exhibit and displays must be related and scientifically accurate, in this case the theme is invertebrates.
    - The displays are mainly created for children.
    - They have to be interactable.
    - Displays shouldn't be too distracting or time consuming.
    - The setting needs bright high contrast colors.
    - It has to have some kind of movement.
    - It can be a game.
    - Multiplayer is a plus but single player is enough.
    - Must not contain text. (Because of language barrier and small kids can't read)
- Ideate: For this the team gathered together and in a round robin fashion we brainstormed ideas, everyone had to participate and every idea was written on the black board. By the end we had a pretty significant pool of ideas to work from, the image attached displays all the ideas. After this everyone picked 3 ideas to work on, the only rules were to pick different ones and they shouldn’t be any of your ideas.



- Prototype: Once everyone had their chosen ideas we started working on them, each prototype had a one hour build time after which we showed our work to our peers and bounced ideas back and forth before starting the next prototype. My three prototypes were:
    - A crab race:
        - Original idea statement: Random crab race.
        - End result: Where the screen is divided horizontally and each player must tap on the target appearing at random on their side of the screen to make the crab move, the first one to cross the screen wins.
        - Already sent to the client.
    - A boat feeding a clam:
        - Original idea statement: Food goes into clam which then opens and releases a pearl.
        - End result: The display shows a boat moving above sea level and a clam at the bottom when the player clicks the screen the boat drops a food pellet in the boat's current position that falls down, if the food gets to the clam it gets fed, after specific number of food eaten the game is won.
        - Already sent to the client.
    - A clam feeding competition:
        - Original idea statement: Divide the screen in two, 2 clams at the bottom and by tapping you create food for them.

- End result: The screen is divided in two vertically and there are two boats above see level and two clams at the bottom, each clam is a player are represented by the color difference. Target appear at random on each half of the screen (color coded) the corresponding player must click on them to drop a food pellet for their clam, after a certain number of click the target repositions itself at random. Once an specified number od food pellets has been eaten by a clam that player wins, the progress of each player is represented by a colored bar at the center of the screen.
- Code below and on the email.

- Test: This step is still pending until our next meeting with the client.

Processing Code

```
/**
*
* Clam feeding competition
* by Hector Ramirez.
*
* The display shows a couple boats over sea level and two clams at the bottom.
* Each player must click on their respective targets as fast as possible to feed the clam.
* The first player to fill the feeding bar wins.
*
*/


boolean gameEnded = false;
boolean p1Won = false;
PVector star = new PVector(10, 30, 5);
PVector targetSize = new PVector(20, 20);
int maxTargetClicks = 10;
PVector clamSize = new PVector(20, 20);
```

```
PVector p1Pos, p2Pos, t1Pos, t2Pos;
PVector foodSize = new PVector(10, 10);
float fallSpeed = 10.0;
PVector boatSize = new PVector(20, 20);
PVector boat1Pos, boat2Pos;
ArrayList<PVector> foodPellets = new ArrayList<PVector>();
int p1FoodEaten = 0;
int p2FoodEaten = 0;
int foodToEat = 20;
int seaHeight = 100;
int barWidth = 10;

void setup() {
  size(640, 360);
  background(0);
  randomizeTarget1Pos();
  randomizeTarget2Pos();
  boat1Pos = new PVector((boatSize.x/2)+(width/4), height-seaHeight);
  boat2Pos = new PVector((boatSize.x/2)+(3*(width/4)), height-seaHeight);
  p1Pos = new PVector((boatSize.x/2)+(width/4), height-(clamSize.y/2));
  p2Pos = new PVector((boatSize.x/2)+(3*(width/4)), height-(clamSize.y/2));
}

void draw() {
  background(0);
```

```
drawScene();


if(gameEnded) {
  if(p1Won) {
    fill(255,0,0);
    star(width/4, star.x+(star.y), star.x, star.y, (int)star.z);
  } else {
    fill(0,0,255);
    star((width/4)*3, star.x+(star.y), star.x, star.y, (int)star.z);
  }
} else {
  fill(255,0,0);
  ellipse(t1Pos.x, t1Pos.y, targetSize.x, targetSize.y);
  fill(0,0,255);
  ellipse(t2Pos.x, t2Pos.y, targetSize.x, targetSize.y);

  fill(255,255,255);
  for (int i = 0; i < foodPellets.size(); i++) {
    PVector tmp = foodPellets.get(i);
      if(tmp.y < height-(clamSize.y/2)) {
        rect(tmp.x, tmp.y, foodSize.x, foodSize.y);
        foodPellets.set(i, new PVector(tmp.x, tmp.y+fallSpeed));
      } else {
        if (tmp.x == boat1Pos.x) {
          p1FoodEaten++;
        } else {
```

```
          p2FoodEaten++;
        }
        foodPellets.remove(i);
      }
    }

    if(p1FoodEaten >= foodToEat) {
      p1Won = true;
      gameEnded = true;
    }
    if(p2FoodEaten >= foodToEat) {
      gameEnded = true;
    }
  }

}

void drawScene() {
  fill(255,255,255);

  rect(0, height-seaHeight, width, 5);

  pushMatrix();
  translate(boat1Pos.x, boat1Pos.y-(boatSize.y/2));
  rotate(-30);
  arc(0, 0, boatSize.x, boatSize.y, -HALF_PI, HALF_PI);
```

```
  popMatrix();


  pushMatrix();
  translate(boat2Pos.x, boat2Pos.y-(boatSize.y/2));
  rotate(-30);
  arc(0, 0, boatSize.x, boatSize.y, -HALF_PI, HALF_PI);
  popMatrix();


  fill(255,0,0);
  rect(width/2-barWidth, height-((height/foodToEat)*p1FoodEaten), barWidth,
((height/foodToEat)*p1FoodEaten));
  pushMatrix();
  translate(p1Pos.x, p1Pos.y);
  rotate(45);
  arc(0, 0, clamSize.x, clamSize.y, -HALF_PI, HALF_PI);
  rotate(45);
  arc(0, 0, clamSize.x, clamSize.y, -HALF_PI, HALF_PI);
  popMatrix();


  fill(0,0,255);
  rect(width/2, height-((height/foodToEat)*p2FoodEaten), barWidth,
((height/foodToEat)*p2FoodEaten));
  pushMatrix();
  translate(p2Pos.x, p2Pos.y);
  rotate(45);
  arc(0, 0, clamSize.x, clamSize.y, -HALF_PI, HALF_PI);
```

```
  rotate(45);
  arc(0, 0, clamSize.x, clamSize.y, -HALF_PI, HALF_PI);
  popMatrix();

  fill(255,255,255);
}

void mousePressed() {
  if(mouseButton == LEFT && !gameEnded) {
    if(checkClick(t1Pos, targetSize, new PVector(mouseX, mouseY))) {
      if (t1Pos.z > 0) {
        foodPellets.add(new PVector(boat1Pos.x, boat1Pos.y));
        t1Pos.z --;
      } else {
        randomizeTarget1Pos();
      }
    }
    if(checkClick(t2Pos, targetSize, new PVector(mouseX, mouseY))) {
      if (t2Pos.z > 0) {
        foodPellets.add(new PVector(boat2Pos.x, boat2Pos.y));
        t2Pos.z --;
      } else {
        randomizeTarget2Pos();
      }
    }
  }
```

```
}

boolean checkClick(PVector oPos, PVector oSize, PVector cPos) {
  if (cPos.x > oPos.x-(oSize.x/2) && cPos.x < oPos.x+(oSize.x/2)) {
    if (cPos.y > oPos.y-(oSize.y/2) && cPos.y < oPos.y+(oSize.y/2)) {
      return true;
    }
  }
  return false;
}

void randomizeTarget1Pos() {
  t1Pos = new PVector(random(targetSize.x/2, (width/2)-(targetSize.x/2)), random(targetSize.y/2, height-(targetSize.y/2)-seaHeight-boatSize.y), random(1, maxTargetClicks));
}

void randomizeTarget2Pos() {
  t2Pos = new PVector(random((width/2)+(targetSize.x/2), width-(targetSize.x/2)), random((height/2)+(targetSize.y/2), height-(targetSize.y/2)-seaHeight-boatSize.y), random(1, maxTargetClicks));
}

void star(float x, float y, float radius1, float radius2, int npoints) {
  float angle = TWO_PI / npoints;
  float halfAngle = angle/2.0;
  beginShape();
  for (float a = 0; a < TWO_PI; a += angle) {
```

```
    float sx = x + cos(a) * radius2;
    float sy = y + sin(a) * radius2;
    vertex(sx, sy);
    sx = x + cos(a+halfAngle) * radius1;
    sy = y + sin(a+halfAngle) * radius1;
    vertex(sx, sy);
  }
  endShape(CLOSE);
}
```