Mini Game Production 2
Technical Design Presentation

Team 3 - The Puppeteers

Procedural Content Generation Overview

- Picture of final game with boxes spawned (any view)
- Picture/gif ground with only boxes spawned (top view)
- Poisson Disk Sampling
    - 2D procedural generated map of points for object (obstacles) placement
    - Easy addition of new obstacle prefabs
    - Easy control of object density in scene
    - Easy control of object size variation
    - O(N) time to generate N Poisson disk samples
    - Citation: *Robert Bridson. 2007. Fast Poisson disk sampling in arbitrary dimensions. In ACM SIGGRAPH 2007 sketches (SIGGRAPH '07). Association for Computing Machinery, New York, NY, USA, 22–es. https://doi.org/10.1145/1278780.1278807*

Procedural Content Generation - Algorithm

Must generate a point list (points have coordinates and rotation values and exist in 2D vector list to pull from when rendering)

- *Find the size of a points' square (in the 2D space) using its given radius (radius = density input)*
- *Determine the number of times the cell size fits into sample region size, for each cell by getting the number of columns and rows (divide the width / cell size and rows / cell size)*
- *Create new vectors of sample candidate points*
- *Put them in a spawn point list (vector2)*
- *While spawn point list is not empty spawn a vector in a random range using* a random *angle of candidate point, new magnitude*
- *Radius/density is min of random range so that candidate is spawned outside spawn center*

Assign this info to candidate point

- *Verify point data in check below*
- Once candidate point is verified, add it as a new spawn point in the list of spawn points to be used in game
- *Record which cell the sample point ends up in*
- *Continue until space is complete (i.e. max rejections reached)*

Check if candidate point can be added into list of points to be spawned in the scene

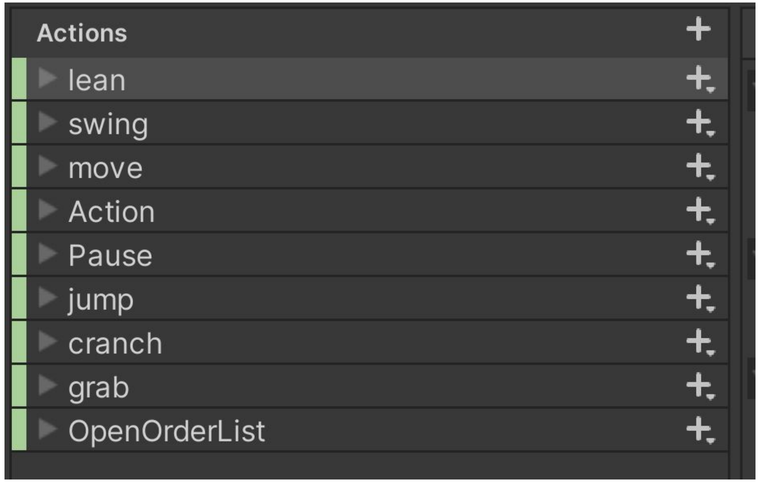
- *Candidate sample vector must be within sample region/zone*
- *Find out which cell the candidate is in, and search surround cells*
- *Get sample point's index*
- *Get distance between point at index and candidate point (using sqrMagnitude bc its cheaper on system to get than mag)*

Conveyor Belt

- Chose a rigidbody conveyor system to affect all objects in scene physically
    - More comical; fits narrative and art style of game
- Normalized vectors for moving objects entering conveyor belts to slow them down enough to stay on the belt (avoiding an overshooting belt entirely)
- Low friction on belt to allow for object movement
- Gif/ picture of object transferring to/from connecting conveyor belts
- Gif/picture of multiple objects on conveyor belt (maybe main belt to show static vs dynamic objects)

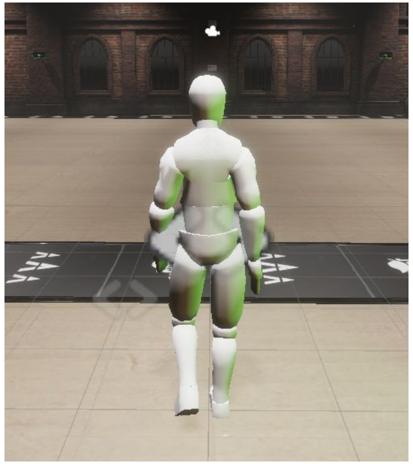
Movement System

- New Unity Input System
- Physics Based Movement System
- Emphasis on unique controller feeling
- Three core movements (+ 3 additional)



Movement System

- Walking Animation applied to "ghost rig" as a walking reference
- Robot rig targets walking reference
- Force applied to mimic less control
- Leaning
  - head moves targeting empty game object with force applied
- Walking
  - head matches ghost head
  - feet target ghost feet
- Arm Control
  - aims at empty game object with force applied

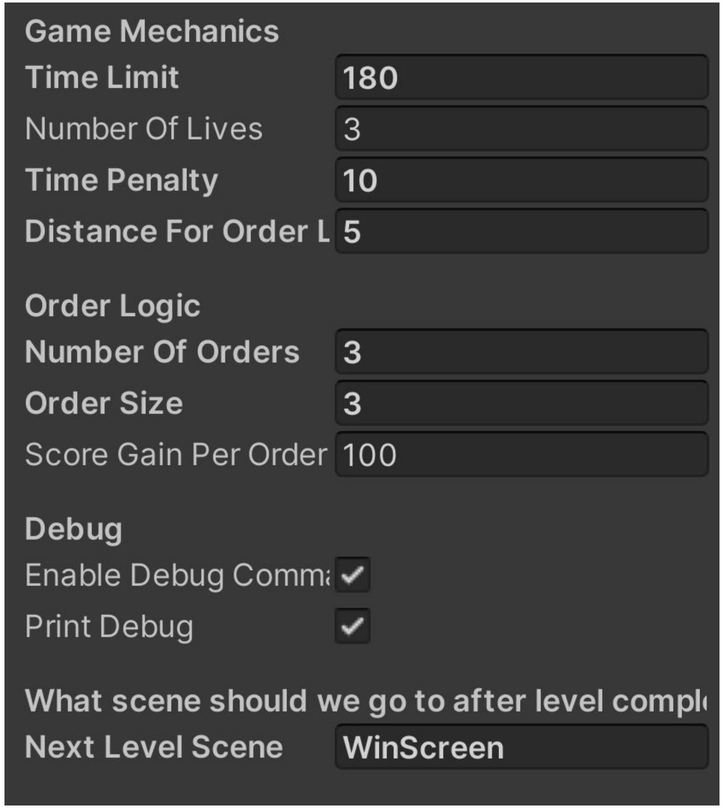


Movement System

- Additional functions related to the movement system
    - Grab
        - simple overview of implementation
    - Crouch
        - simple overview of implementation
    - Jump
        - simple overview of implementation

Game Manager

- Serialized game logic for game and level designers
- Centralized logic used in all game levels
- Communication between other scripts
- Interacts with the conveyor belt and order manager to generate and ship orders
- Responsible for integration of save/load into levels



Game Mechanics

Time Limit	**180**
Number Of Lives	3
Time Penalty	**10**
Distance For Order L	**5**

Order Logic

Number Of Orders	**3**
Order Size	**3**
Score Gain Per Order	100

Debug

Enable Debug Comma	✓
Print Debug	✓

What scene should we go to after level compl

Next Level Scene	**WinScreen**

Save and Load

- Checkpoint saving system
    - Saving at the beginning of each level
- No saving during the level, only at the last checkpoint (start of current level)
- Orders, shelves, and obstacles are not saved due to the destructive nature of the game
    - Unable to save the game in an unplayable state
- Upon load, will start level from the beginning of the level they were on at save

UI Management

- UI manager is responsible for updating all UI elements on the screen
  - score
  - orders
  - number of lives
- Gets all necessary updates from Game Manager
- Does not have a monobehavior update function
  - all updates manually called by game manager

Order Manager

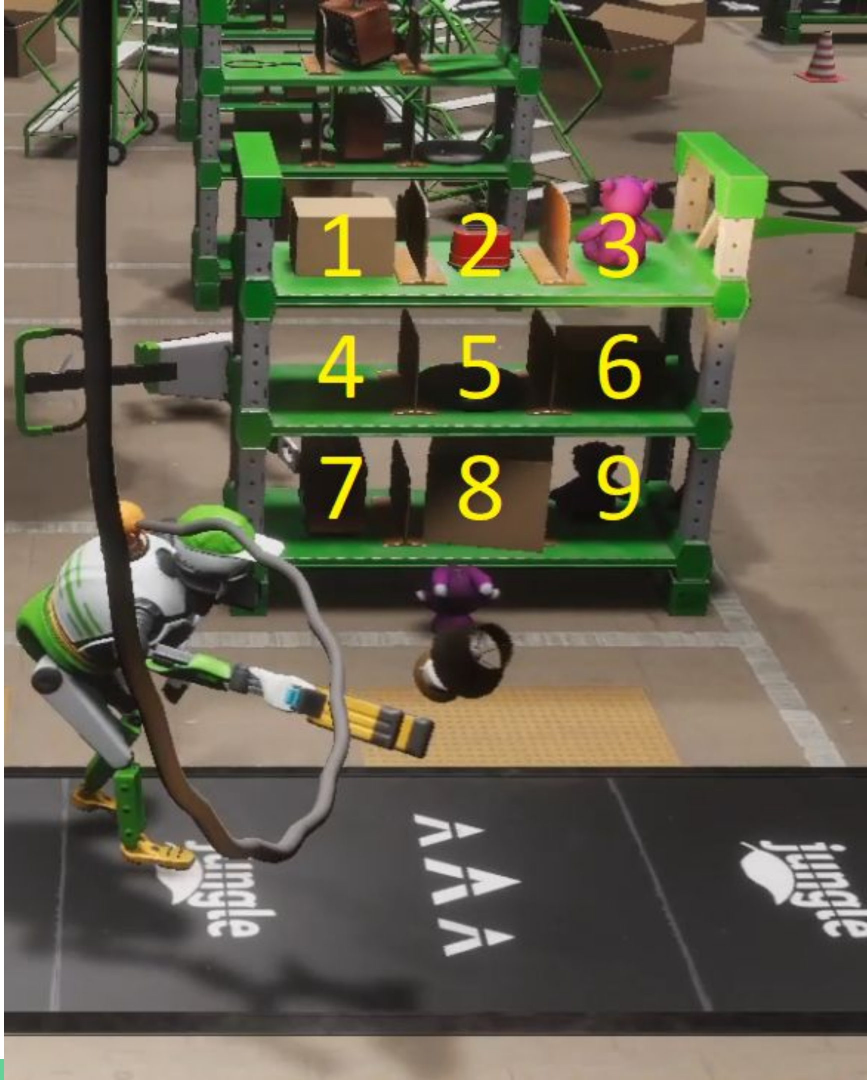
- Responsible for the logic of creating and fulfilling orders
- Uses list of grabbable assets in scene to create order
    - ensures all orders are completable at the time of their creation
- Removes items from order upon request from game manager
- Notifies game manager when an order is complete

Shelf Item Spawning

- GameManager finds all shelf-tags in scene (achieved via “shelf”-tags)

- The shelves are then populated randomly by assets from a list of “Grabbable” assets and “Decor” assets

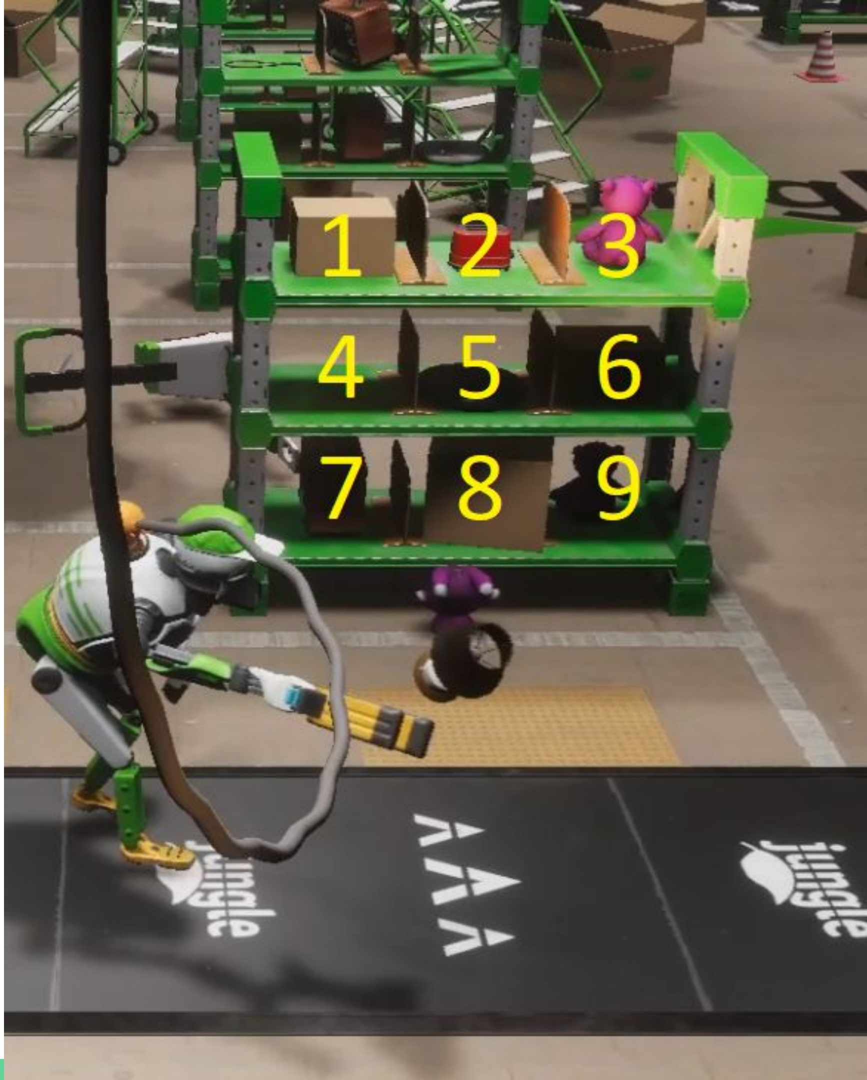
- Manually configurable probability of generating:
    - a “Grabbable” asset
    - a “Decor” asset
    - an empty asset



Shelf Item Spawning

- A min and max amount of a "Grabbable" assets on each shelf and scene is also configurable

- The number of assets generated are tracked;
  - If max "Grabbable" assets are reached, prevent further spawns
  - If min "Grabbable" assets won't be reached, Forcefully spawn in a "Grabbable" asset on remaining spots


1
2
3
4
5
6
7
8
9
jungle
jungle