COLLEGE OF ENGINEERING

Electrical Engineering and Computer Science

Monogame Party 2018

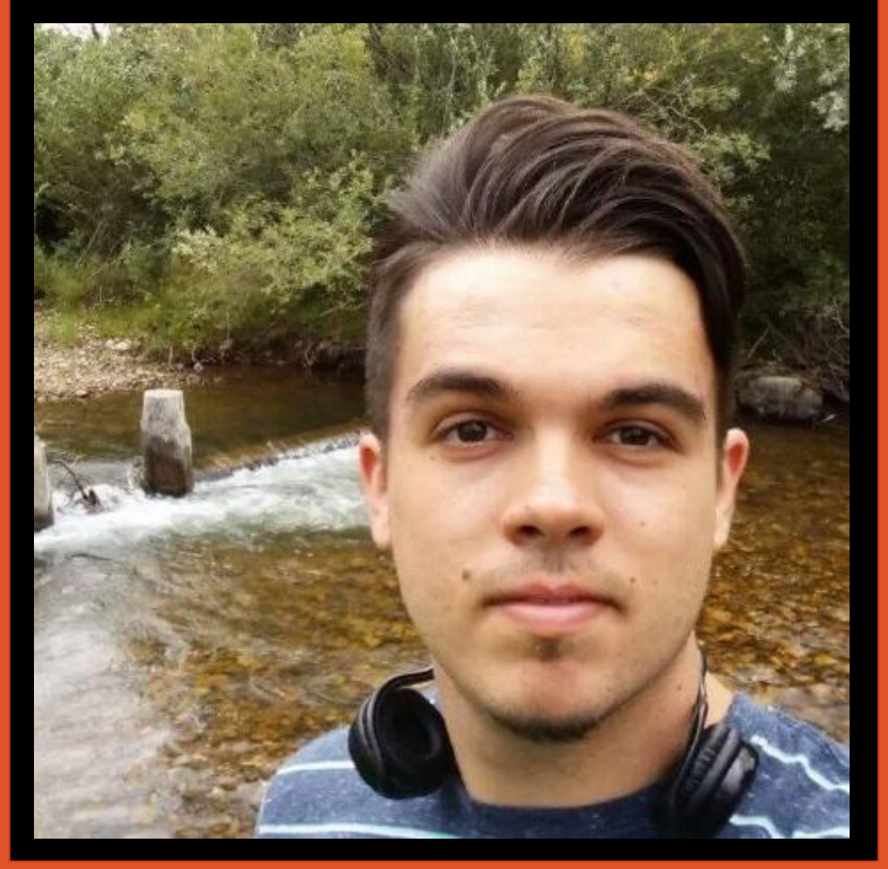
A Mario Party inspired video board game experience for up to two human and two to three scripted AI players. Traverse the dangerous waters and beaches of Pirate Bay! Collect coins, purchase stars and battle your foes to become the richest meeple that ever dared set anchor in this menacing harbor!

Title Screen Artwork

Team Caelum

Meet the Team

James 'Cam' Abreu



- Game Engine
- Board 'spaces'
- Board Graphics
- Music/SFX
- Audio Engine
- Chance Time

Chris Bugsh

- Menu System
- 'Rounds' Logic
- AI Scripting
- Purchasing Stars
- Meeple Movement
- Minigame 1

Phillip Jarrett

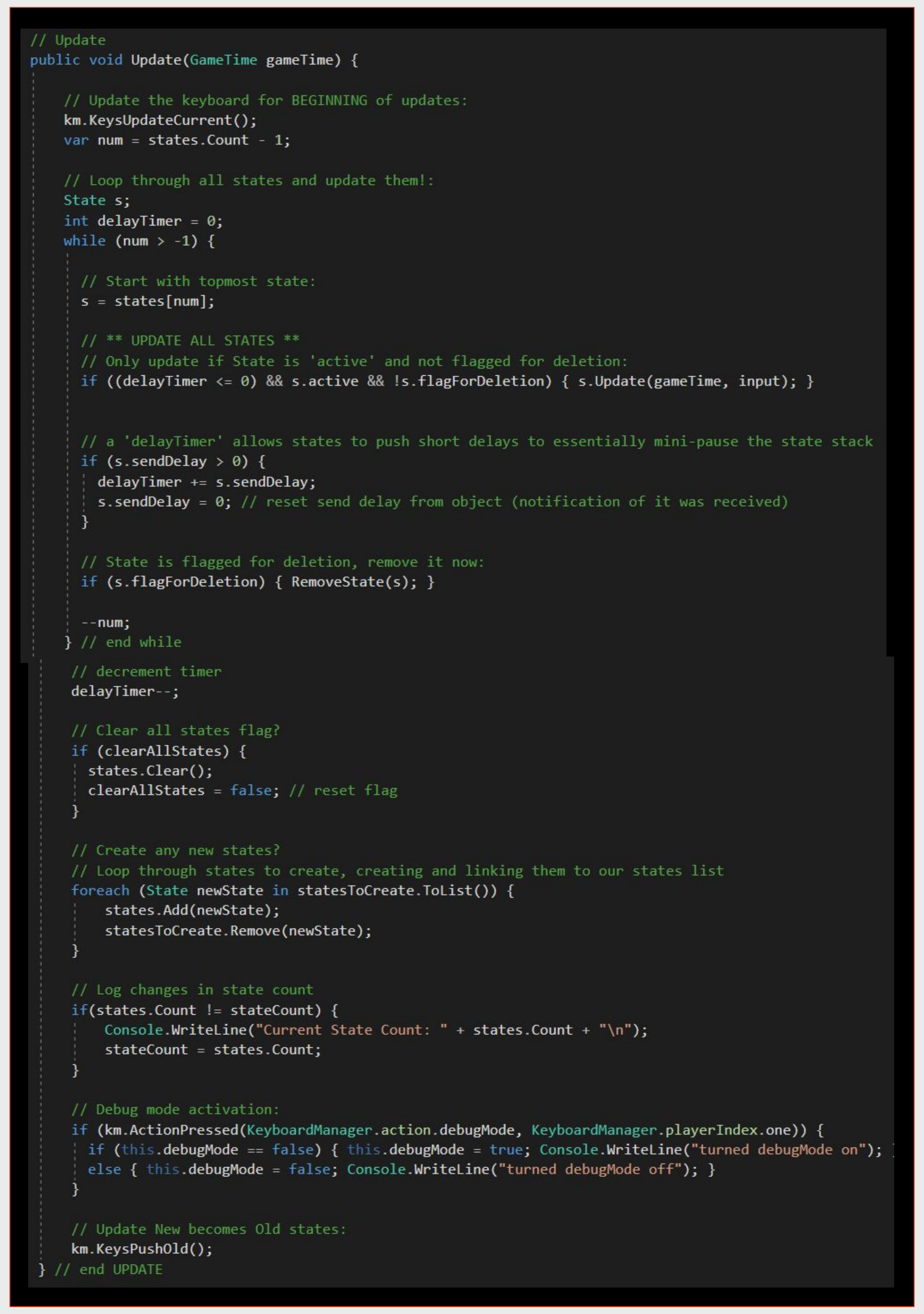
- Repository Lead
- Dice Block
- Minigame 2
- Testing / Debugging

Oregon State University

Game Engine

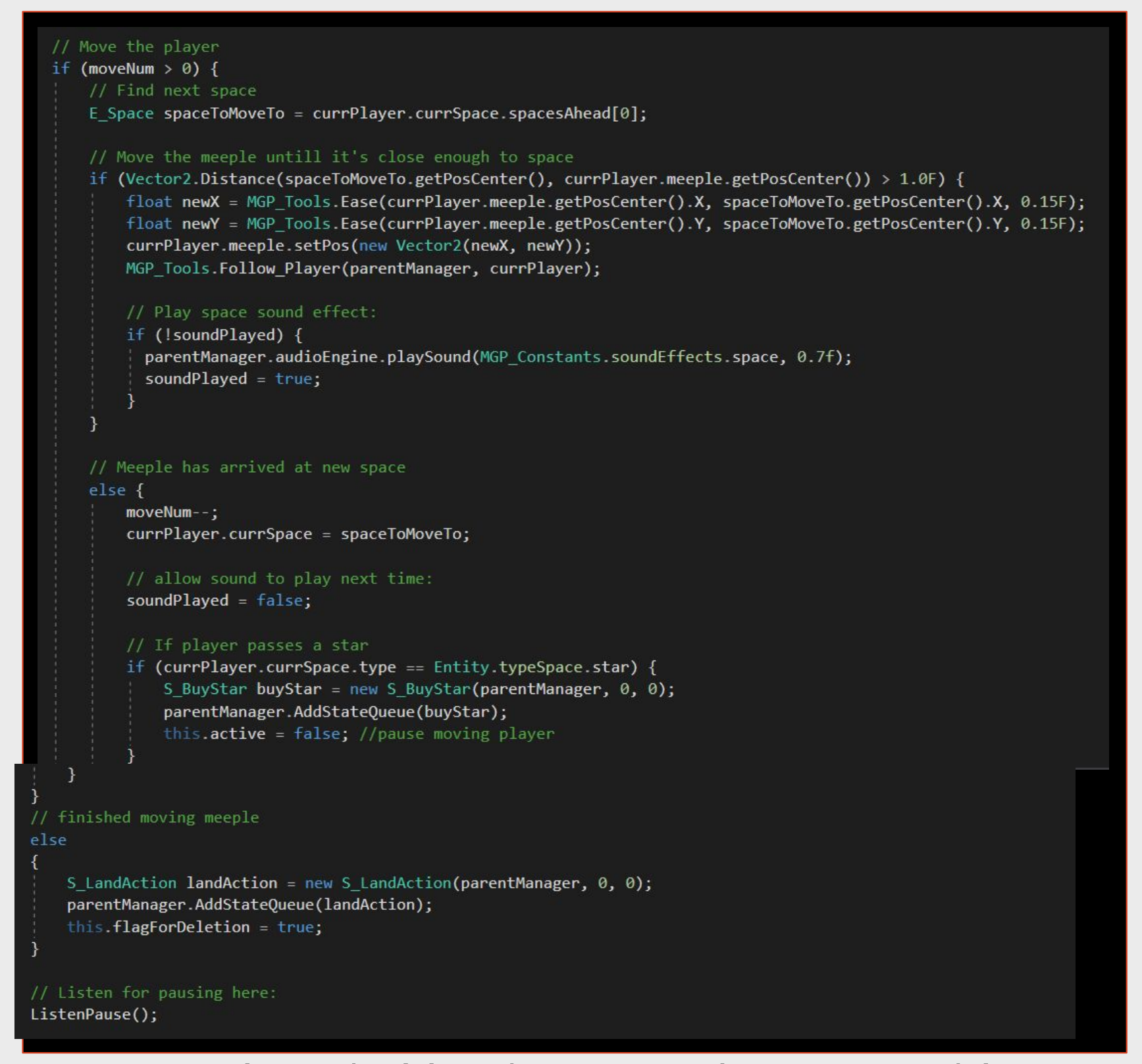
- C#, Visual Studio, Monogame
- Cross Platform Release
- Game State Manager
- States
- Audio Engine

Like many game engines, a custom stack of game 'states' runs the core of the game. A 'Game State Manager' organizes, destroys, pauses, states as needed. See Figures 1a and 1b (right):



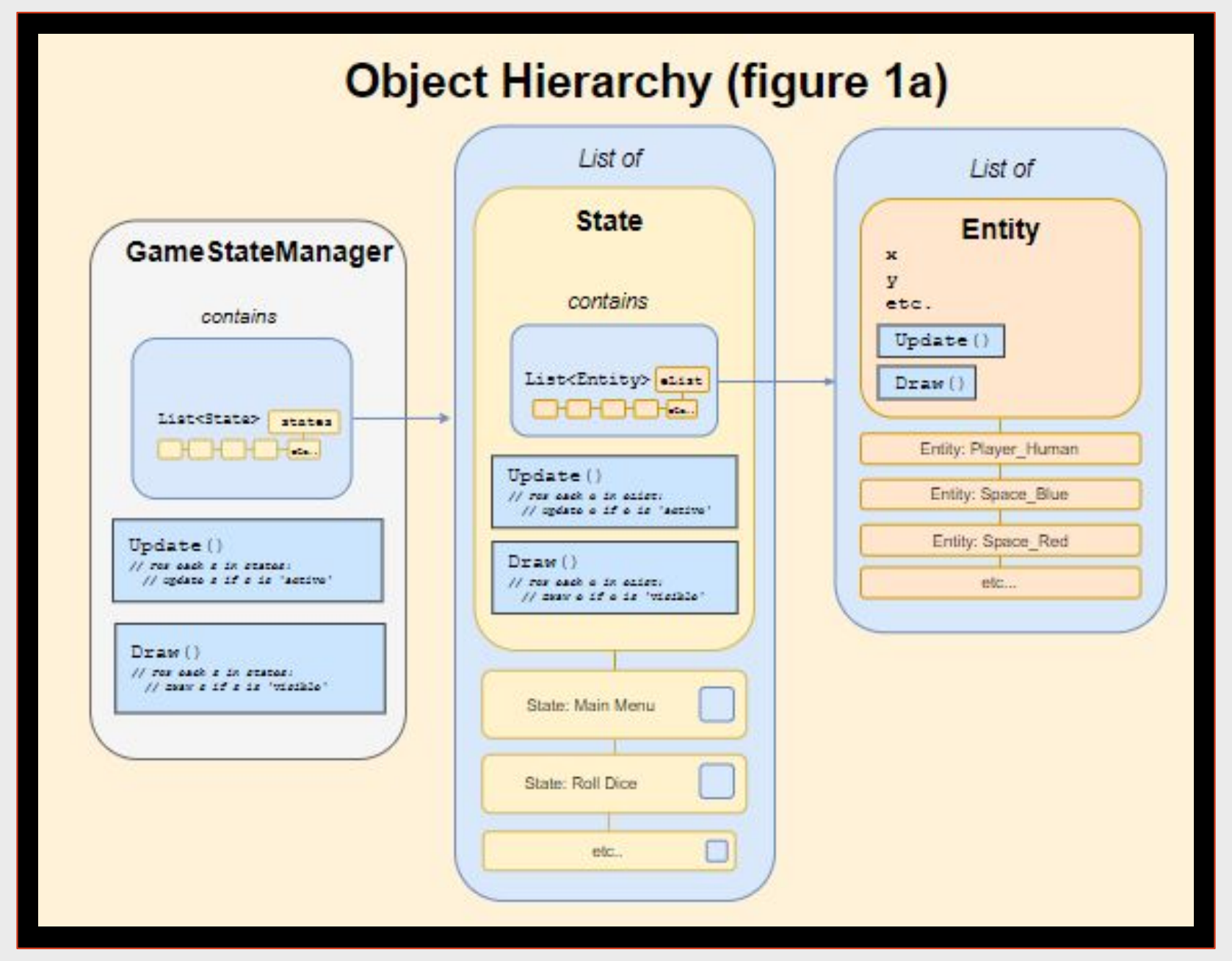
```
// Update
public void Update(GameTime gameTime) {

    // Update the keyboard for BEGINNING of updates:
    km.KeysUpdateCurrent();
    var num = states.Count - 1;

    // Loop through all states and update them!:
    State s;
    int delayTimer = 0;
    while (num > -1) {

      // Start with topmost state:
      s = states[num];

      // ** UPDATE ALL STATES **
      // Only update if State is 'active' and not flagged for deletion:
      if ((delayTimer <= 0) && s.active && !s.flagForDeletion) { s.Update(gameTime, input); }


      // a 'delayTimer' allows states to push short delays to essentially mini-pause the state stack
      if (s.sendDelay > 0) {
        delayTimer += s.sendDelay;
        s.sendDelay = 0; // reset send delay from object (notification of it was received)
      }

      // State is flagged for deletion, remove it now:
      if (s.flagForDeletion) { RemoveState(s); }

      --num;
    } // end while
     // decrement timer
     delayTimer--;

     // Clear all states flag?
     if (clearAllStates) {
       states.Clear();
       clearAllStates = false; // reset flag
     }

     // Create any new states?
     // Loop through states to create, creating and linking them to our states list
     foreach (State newState in statesToCreate.ToList()) {
         states.Add(newState);
         statesToCreate.Remove(newState);
     }

     // Log changes in state count
     if(states.Count != stateCount) {
         Console.WriteLine("Current State Count: " + states.Count + "\n");
         stateCount = states.Count;
     }

     // Debug mode activation:
     if (km.ActionPressed(KeyboardManager.action.debugMode, KeyboardManager.playerIndex.one)) {
       if (this.debugMode == false) { this.debugMode = true; Console.WriteLine("turned debugMode on"); }
       else { this.debugMode = false; Console.WriteLine("turned debugMode off"); }
     }

     // Update New becomes Old states:
     km.KeysPushOld();
} // end UPDATE
```

'Update' code in Game State Manager

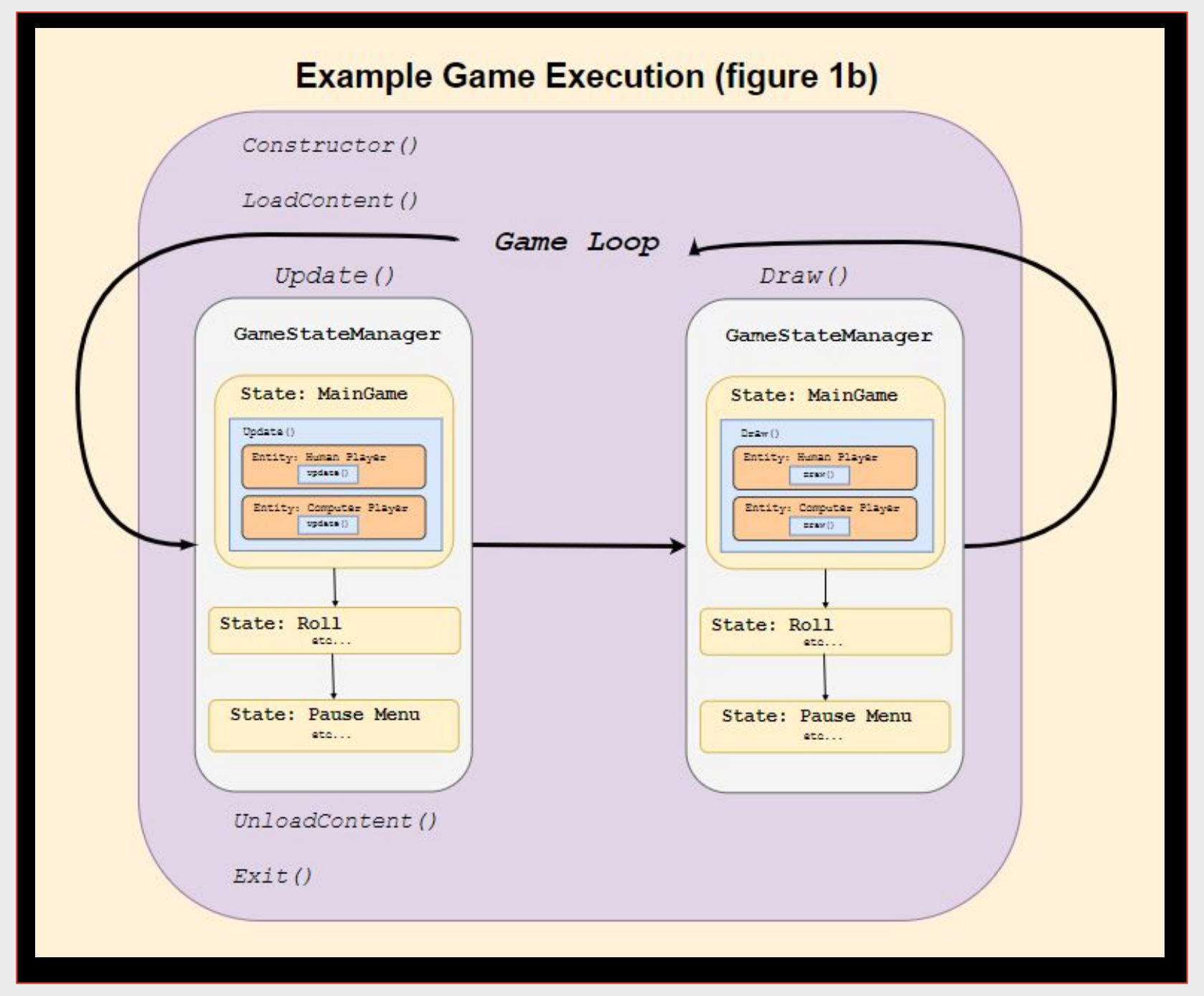


```
// Move the player
if (moveNum > 0) {
    // Find next space
    E_Space spaceToMoveTo = currPlayer.currSpace.spacesAhead[0];

    // Move the meeple untill it's close enough to space
    if (Vector2.Distance(spaceToMoveTo.getPosCenter(), currPlayer.meeple.getPosCenter()) > 1.0F) {
        float newX = MGP_Tools.Ease(currPlayer.meeple.getPosCenter().X, spaceToMoveTo.getPosCenter().X, 0.15F);
        float newY = MGP_Tools.Ease(currPlayer.meeple.getPosCenter().Y, spaceToMoveTo.getPosCenter().Y, 0.15F);
        currPlayer.meeple.setPos(new Vector2(newX, newY));
        MGP_Tools.Follow_Player(parentManager, currPlayer);

        // Play space sound effect:
        if (!soundPlayed) {
          parentManager.audioEngine.playSound(MGP_Constants.soundEffects.space, 0.7f);
          soundPlayed = true;
        }
    }

    // Meeple has arrived at new space
    else {
        moveNum--;
        currPlayer.currSpace = spaceToMoveTo;

        // allow sound to play next time:
        soundPlayed = false;

        // If player passes a star
        if (currPlayer.currSpace.type == Entity.typeSpace.star) {
            S_BuyStar buyStar = new S_BuyStar(parentManager, 0, 0);
            parentManager.AddStateQueue(buyStar);
            this.active = false; //pause moving player
        }
    }
}
// finished moving meeple
else
{
    S_LandAction landAction = new S_LandAction(parentManager, 0, 0);
    parentManager.AddStateQueue(landAction);
    this.flagForDeletion = true;
}

// Listen for pausing here:
ListenPause();
```

How 'Meeples' (or player pieces) move in code'

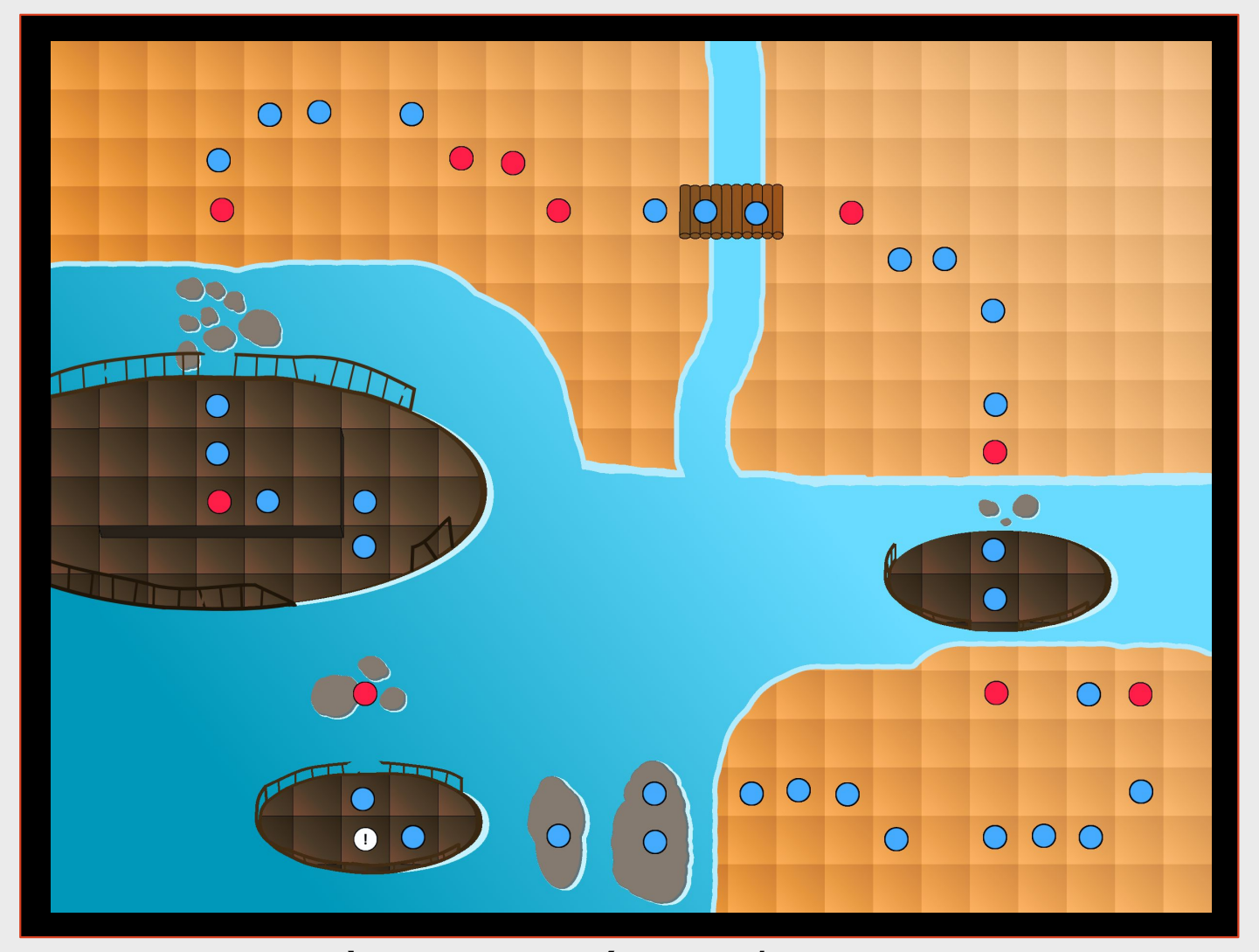


Note: Pressing F2 while in game will enable debug mode, which will show the current stack of game states.

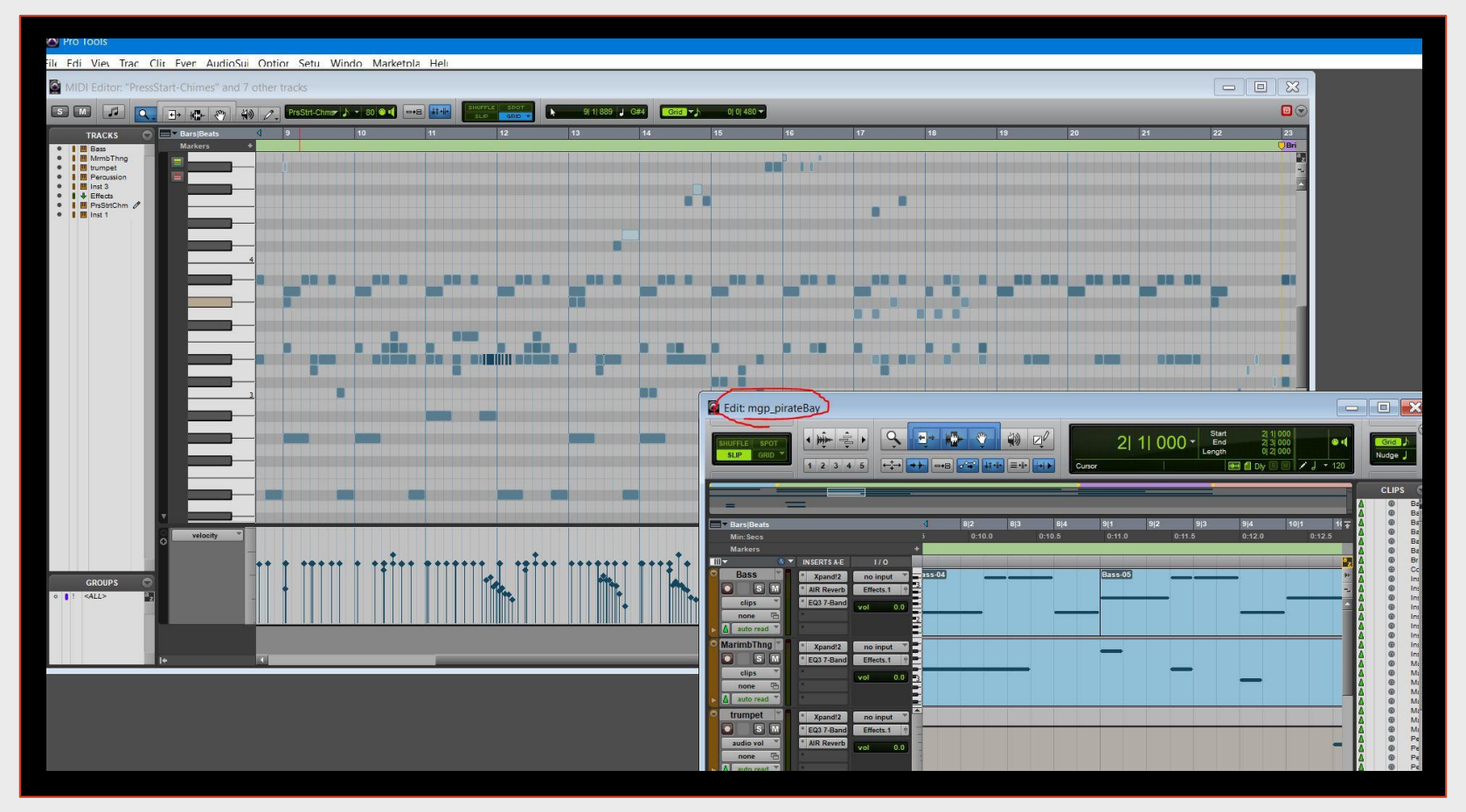


Custom Created Assets

- All graphics used in the game itself were created by the Caelum team.
- Music and SFX were composed entirely by James Abreu
- Custom title screen artwork was created by Janice Dixon



'Pirate Bay' Board Design



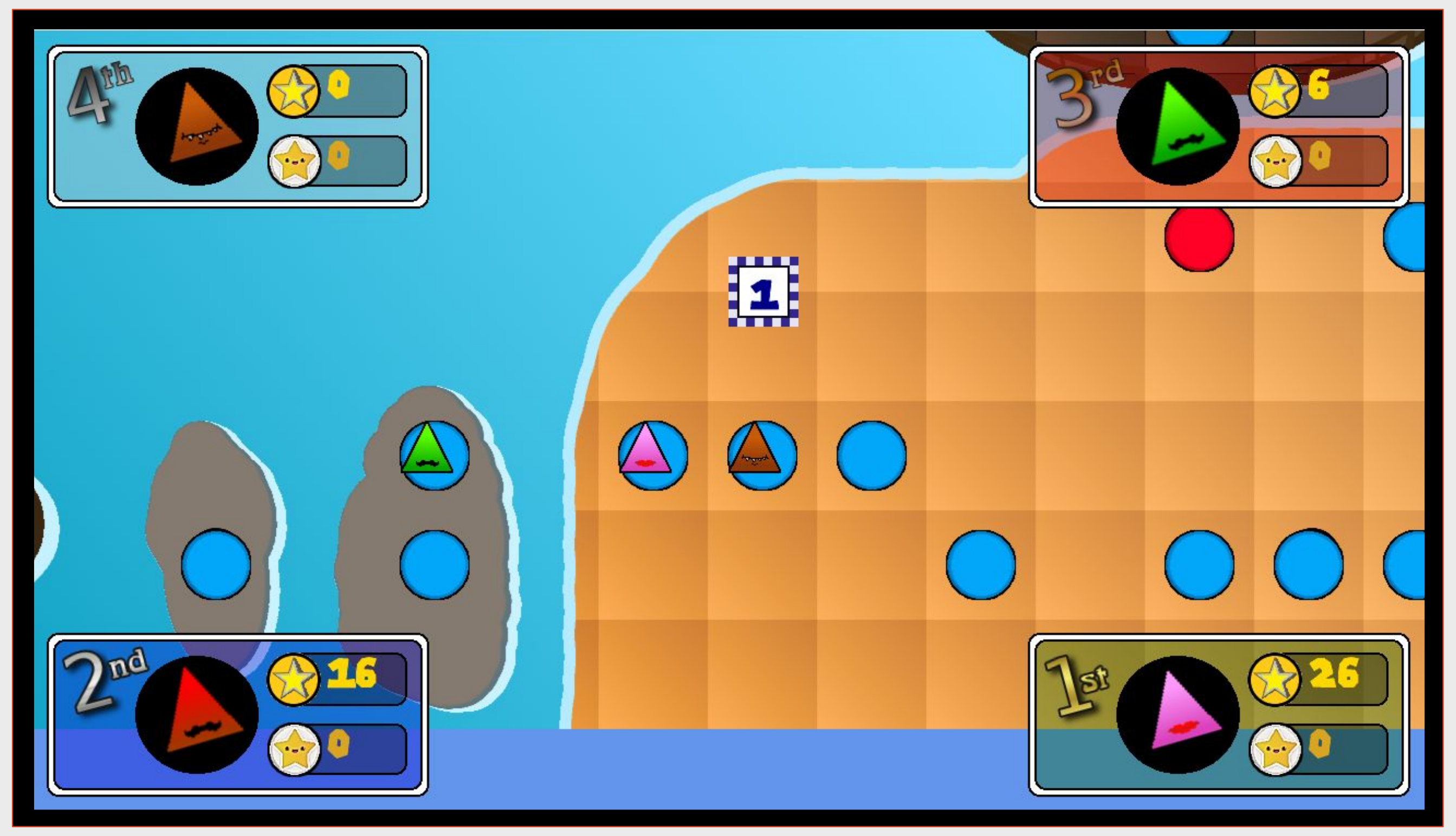
Audio Assets Composed in Pro Tools 11

Game Features

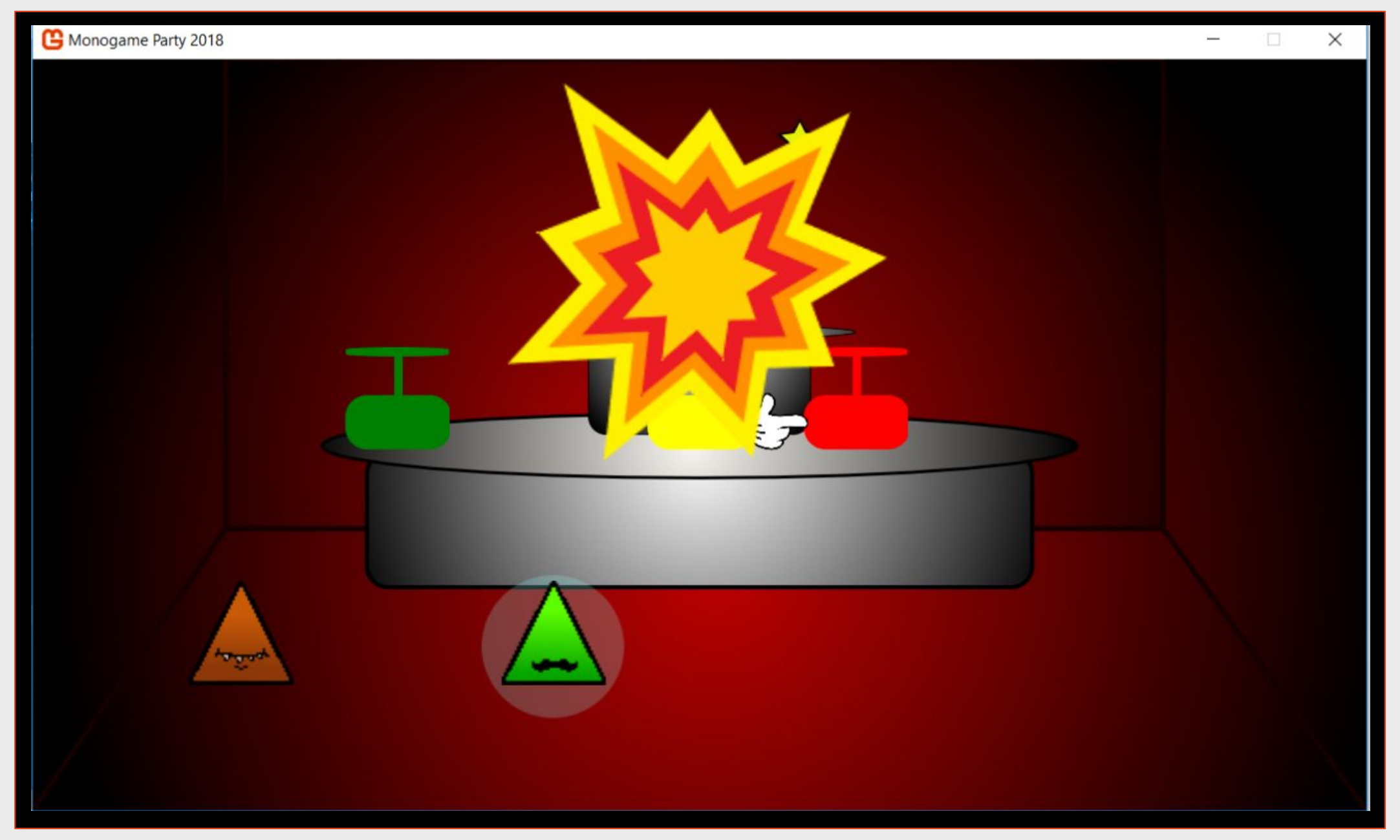
- Choose from six different characters
- Three different A.I. difficulties
- Three different options for game length
- Choose whether or not to award bonuses at the end of the game
- Animated dice rolling
- Animations for landing on different spaces and buying stars
- Alternating selection of mini games to play

https://github.com/jtrain184/mgp18

Gameplay Screenshots



Rolling the die



Don't blow the bomb! *Minigame 1*