**Blackout Bingo**

## Barak Katz

## Overview

The purpose of this document is to describe some technical and architectural aspects of the test related game. It is not intended to be a full Game Design Document (GDD) or a Technical Design Document (TDD).

The game is a clone of ”Blackout Bingo”, demonstrating Unity and design capabilities. The game is designed to be as generic as possible. Which would allow for instant changes in almost all number related elements (number of tiles presented on the board, balls numeric range etc.), allowing a relatively quick adjustment, thus adding flexibility, or to create a similar game aimed at a different target audience, such as in the case of “British Bingo”.

# Concept

An online competitive bingo which adds **reflexes** and **planning** as the keys to winning.

# Features

These features describe the Minimum Viable Product (MVP) as I see it, as well as the introduction of some additional features which may enhance the player’s experience.

As in bingo, each player is handled a board of **NXN** tiles, filled with numbers within a certain range **Y** of integers. After every **X** number of seconds, a random new number from the range is pulled out and displayed. After a certain **T** time passes, the game ends.

For the current implementation, their values are: N=5, Y=75, X=3, T=75

Each player can select any of the numbers she has on the board at any time. If the number is matching the last number pulled from the **Y** range (or a recently pulled one), she may score additional points, based on the time it took her to do so (faster is better). In addition, a streak of correct clicks helps fill a meter which allows some special powerups, who would help the player score higher. Scores are also achieved by having a diagonal, horizontal or vertical full rows (Bingo).

Currently, for the simplicity of implementation, the server supplies the ongoing information for the pulled balls to all players, and expects the client to respond only on the game’s end. The client terminates the game on one of the following conditions: Either local time expired, or the Remote server has no more balls to pull from.

# Environments and Tools

The software should be developed using the following environments and tools:

* Unity based game (Using C# as the main programming language).
* Github (Git) extension for version control.
* PUN2 (Photon) for online functionality.
* TMP (Text Mesh Pro) for better looking UI elements (official Unity package).

# Test Plan

Check locally and offline the core game mechanics - Done

Check with a server plus one player the core game mechanics - Done

Check with a server plus two simultaneous players the core game mechanics – Done (with 3)  
  
core game mechanics includes –

* Clicking on tiles functionality
* Timer functionality
* Score correctness (Time related and Bingo related)
* Bingo detection and marking
* Audio functionality (action related and background music)
* Game Over state

Synchronization between the different players – can be demonstrated with the audio synchronization.

Out of Scope – Tests which are left for future work.

* Server / Player Crash / Log off.

Expected result – should not crash the game for the other players. At most it could be a momentary delay in the gameplay.

## Design

Enabling flexibility and control, the chosen architecture is client-server based, where the server can serve a number of players, sharing the same round with different cards but identical pulled balls sequence.

# Client-Server architecture

The main two entities (or actors) are the **server side (host)**, which would be called the “**Remote Game Manager**”, and the **player side (player)**, which would be called the “**Local Game Manager**”.

Due to a Photon limitation, and to maintain a single source code for all, each client has them both in scene, however, only the related actions are accessible. At the start of the scene, the PlayerSpawn gameobject sets the first logged in player to be the host, and all others to be the players.

The following figure describes the basic modules contained in each type of these entities (Server or local player).

On the server side, play initialization includes setting the possible ball numbers range, and creating each player’s bingo card, to allow sending for each local game manager. Once all players are ready to begin, the In-game data generation starts creating the sequence of pulled balls and sends them one at time, simultaneously to all players. Upon termination condition, the Remote game manger’s Play termination, verifies that all players had reached the end game state. Once all players have sent their end game’s statistics, the Remote game manager Post round activity calculates the winner and send the corresponding data to the local game managers and should save the rest for future analysis.

On the player side, Init player and game setup creates the local board with the given data from the Remote game manager, including the initial score of the round and the timer preset value. Player inputs and game execution is the main play subtask. It uses the remotes game manager’s In-game data generation to run the game, as well as the local player’s input. It is the local timer responsibility for declaring a local end game state. Whether it was declared locally or received by the Remote game manager, upon reaching the end game state the Local game manager should send all statistics and game data to the Remote game manager.

## Gameplay

The gameplay should be similar to the game “Blackout Bingo”.

# Initialization

Load a start screen with a card, and an empty power meter. Scores are initialized to zero for a single session game, and per host decision, could be related to past games in a multi-session.

# In-game Scoring

Points are earned in two ways:

* **Clicking on rightfully pulled balls** scores points, according to the next (generic) ledger:
  + Very Fast (less than 0.5 second) = 5\*scoreValue, where scoreValue=100.
  + Fast (less than 1 second) = 3\*scoreValue
  + Ok (less than 2 seconds) = 2\*scoreValue
  + Slow (less than 3 seconds) = 1\*scoreValue.
  + Any slower clicking would still mark the tile but would not add points.
* For the implemented value:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time from pull to click [sec] | 0<t<=0.5 | 0.5<t<=1 | 1<t<=2 | 2<t<=3 | 3<t |
| Points Added | 500 | 300 | 200 | 100 | 0 |

* **Scoring a bingo** (a full row of rightfully clicked tiles, horizontally, vertically or diagonally). All new bingos are calculated only after pressing the bingo button, and add points according to the next formula (if the number of bingos is 1 and above):
  + (2^(number of bingos + 1))\* bingoValue, where bingoValue=1000

For example:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of new Bingos | 0 | 1 | 2 | 3 | 4 |
| Points Added | 0 | 4000 | 8000 | 16000 | 32000 |

Powerup abilities:

* There are two different implemented powerups, with the generic implementation to add multiple others (made with a similar design to Strategy design pattern).
  + Time = adds 10 seconds to the local player’s timer
  + X2 = for the next 10 seconds, all points earned are doubled (powerup is accumulated).
* Powerup gains are determined by the speed of the correct clicking as in with the points, with the following rates:

VeryFast = 100, Fast = 60, Ok = 30, Slow = 15 (out of a filled meter of 100).

* In addition, the powerup accumliation is reduced by 25 for clicking on an incorrect tile.

# Termination

On the game’s end, the above score calculation is sent to the Remote game manager. Optionally, all relevant game data could be sent for future analysis.

## Art

# Art Style

As in the original game, the game should have a combination of light cyan and light magenta, which are calming colors.

# UI

Most of the UI is created on runtime, which allows the most dynamic game changes. Could be done in advance to look nicer.

Pulled balls have a color indicating their corresponding column.

The position of the buttons could ease transferring into mobile, as most of the action is in the bottom center of the screen, with the two buttons being on opposite sides fitting two thumbs. However, for the purpose of this test, I assume that it would be tested on a PC screen.

Note: The larger last pulled ball was not yet implemented.

# Out of Scope

The following features are good to have but are not planned for the current implementation.

* Security/anti cheaters measures.
* Offline / connectivity issues handling and full testing.
* Limited to 20 users only at most on the application (free Photon limitation)
* The center number should be free from the beginning, it is right now a standard bingo tile.
* The most recently pulled ball is not bigger or marked differently than the other previously drawn ones.
* Action should have a more animated feedback, given the time constraints this was considered at a lower priority.
* Although the board size and most UI elements are flexible and are generated on runtime, some UI elements such as the size of Tiles are predetermined, in order to ensure number readability and location within the tile.
* A UML diagram could have upgraded this document, however due to time limitations it wasn’t made.
* Should consider implementing an algorithm that verifies the player receives higher chances of receiving a useful ball, in order to keep the players engaged and happy with their play (based on reactions of watching players playing with my implementation).
* The UI elements and overall look could be improved, especially the sharpness of the UI elements on a full screen.

# Actions

Pregame Server –

Initiate the game when enough players have joined

Game manager / Server –  
Generate random game boards

Pull a new number and display it  
Display scores by the end of the game

Pregame player –

Allow name change

Join game

Player –

Click to earn points

Click on possible bingo numbers

Declare a Bingo

Fill a progress bar

Use a powerup  
Announce a game over if all numbers were picked up, or if local time ran out  
  
Bingo Number –

Be clicked – if is valid score points, fill meter etc.

Indicate status – not used, clicked correctly, clicked incorrectly (and return to not used)  
  
Timer -  
Show the current player’s time left for the game  
  
Score Display –

Format points and display them  
  
Ball –

Holds value and time pulled

# Screens

The following screens are proposed for a full version of the game. However, due to time limitations and priorities only some of them are implemented.

Server Main menu

Player Main menu

Player in game board

Player special power up screen

End game scores

# Controls

## The game is mouse based only. Audio

# Sound Effects

The game should indicate on the following actions. The following actions –

* Clicking on a rightfully selected tile.
* Clicking on a wrongfully selected tile, either already selected, or not yet called.
* Scoring a bingo
* Using a powerup
* Filling the powerup meter
* Game over

# Music

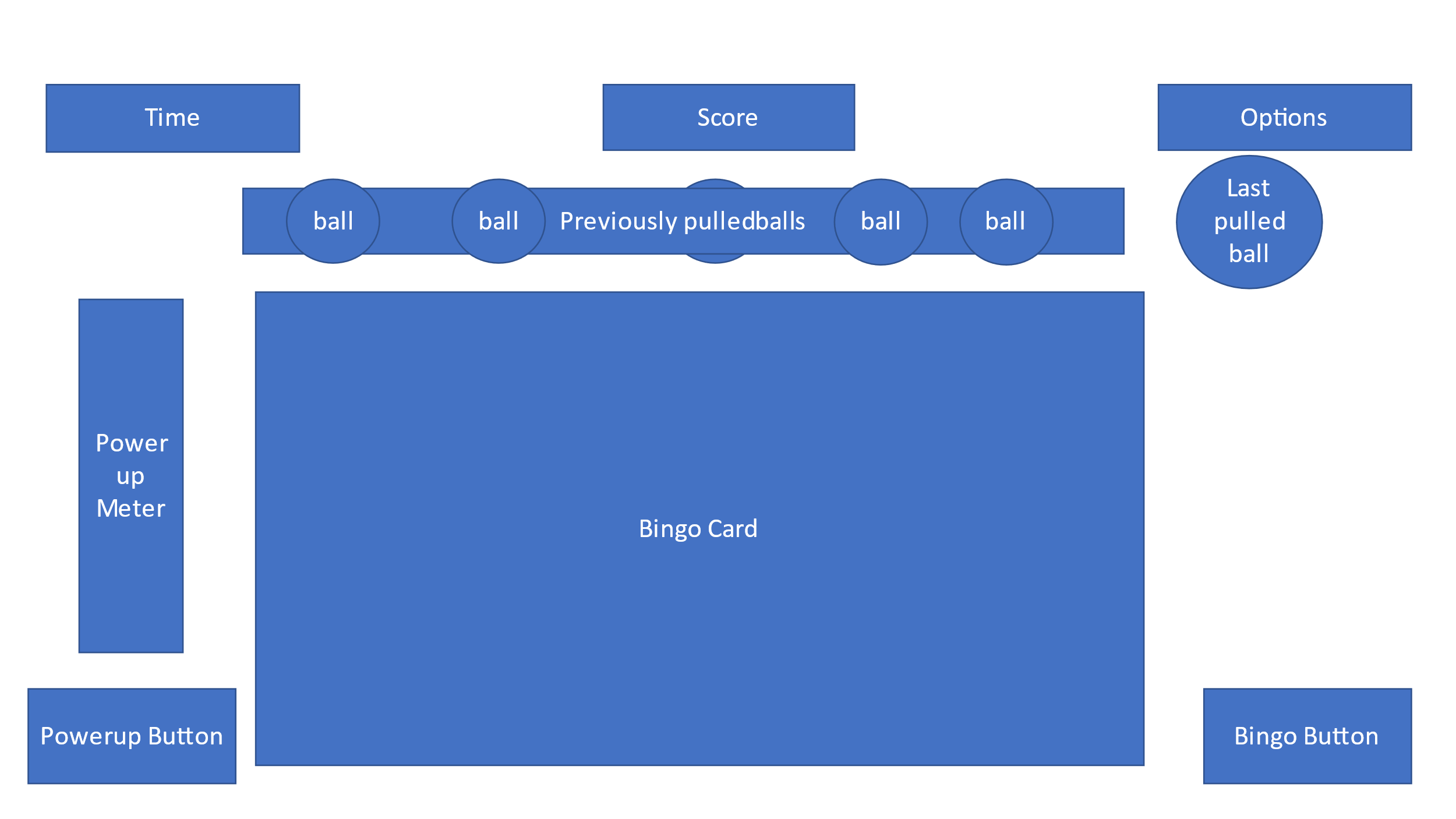
Should add a quitter (compared to the sound effects) background music.

## Gameplay Diagram

Following is an initial location of the main game objects.

# Screen Layout

The guideline screen to be aimed for is as the following.



## Work Plan

The following is the initial proposition for work segmentation, as was mostly followed to the letter on the git repository. Since this is an extremely short time project, not timing information was planned in advance, and some of the optional propositions were delayed and may not be included on submission.

# 1.Basic Player interaction – Have a timer with a popup. A score is based on the speed of the click. (Limited UI)

Timer -  
Show the current player’s time left for the game

Score Display –

Format points and display them

Ball –

Holds value and time pulled

Player –

Click to earn points

# 2.Bingo board creator, with pulling new numbers and displaying them

Player in game board

Game manager / Server –  
Generate random game boards

Pull a new number and display it

# 3.Combining the two and scoring only when the desired number is clicked.

End game scores

Bingo Number –

Indicate status – not used, clicked correctly, clicked incorrectly (and return to not used)

Player –

Click on possible bingo numbers

Announce a game over if all numbers were picked up, or if local time ran out

Fill a progress bar

Bingo Number –

Be clicked

Game manager / Server –  
Display scores by the end of the game

# 4.Adding Bingo declaration functionality.

Player –

Declare a Bingo  
Add Visual Cues on Bingos

# 5.Separation into Server and Clients

Server Main menu

Player Main menu

Pregame Server –

Initiate the game when enough players have joined

Pregame player –

Allow name change

Join game

# 6.Allow score progression

# 7.Add Power ups

Player special power up screen

Player –

Meter and filling it

Use a powerup

8.Improve final look, including ui and audio.  
Added audio effects and background music

## Actual Product Samples

Some representing images from the game:  
  
  
  
  
  
And a behind the scenes look:  
