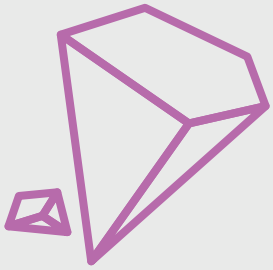


REACTIVE SPACES API

DESIGN DOCUMENT

DESIGN STUDIO 4 | NOV 7 2014

FOR PROF. ALI ARYA | MARCO | MATTHEW | RYAN | ZARA

TABLE OF CONTENTS

1 INTRODUCTION

1.1 HIGH CONCEPT

Reactive Spaces is an API which allows for easy development of distributed javascript web apps that utilize the Microsoft Kinect.

1.2 API OVERVIEW

The API will consist of a desktop application, a server, and a JavaScript (JS) Library. The desktop app will manage a socket connection to the server and expose a local websocket connection over which to send and receive data with the JS library. It will be responsible for passing messages to and from the server and JS library as well as reading from and passing Kinect input data. The JS library will deal with connecting to and sending & receiving data from the desktop app. It will also define data types specific to the API as well as providing utility functions related to those data types. The server will then manage connections between all running instances of the desktop app, and match them into sessions based on app types.The server will also be responsible for passing messages between appropriate clients in each session.

1.3 GAME OVERVIEW

The game will aim to demonstrate all the features of the created API. The game will involve multiple stations participating and playing against each other. The game will be an arcade-style game where players aim to reach a high score. The game will use the API to provide input via the Microsoft Kinect. It will also use the API to share data between peers. The game will be influenced by the other stations who are also running the game.

2 API

2.1 GOALS

- Expose local kinect interaction data through a JS library
- Expose remote kinect interaction data through a JS library
- Allow for custom data to be sent between game instances
- Match and organize connections between game instances

2.2 SPECIFICATIONS

2.2.1 DESKTOP APP

PLATFORM / REQUIREMENTS

- Windows Vista or greater with .NET 4.5
- Kinect 360 sensor and Kinect runtime or SDK v1.8
- Internet connection

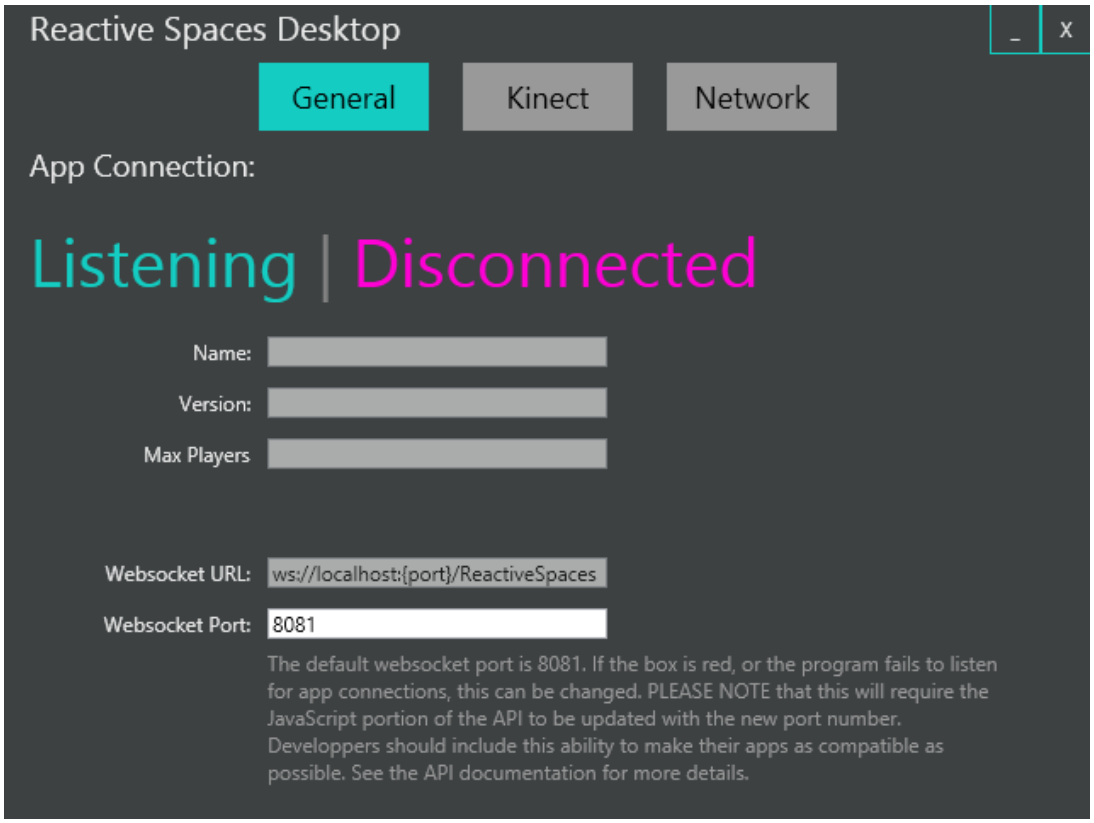
EXPOSED FUNCTIONALITY

- Set name and location information for the station
- See Kinect data and sensor status
- View connected application information and connection status
- View server connection status and information about stations in the same app session

INTERFACE

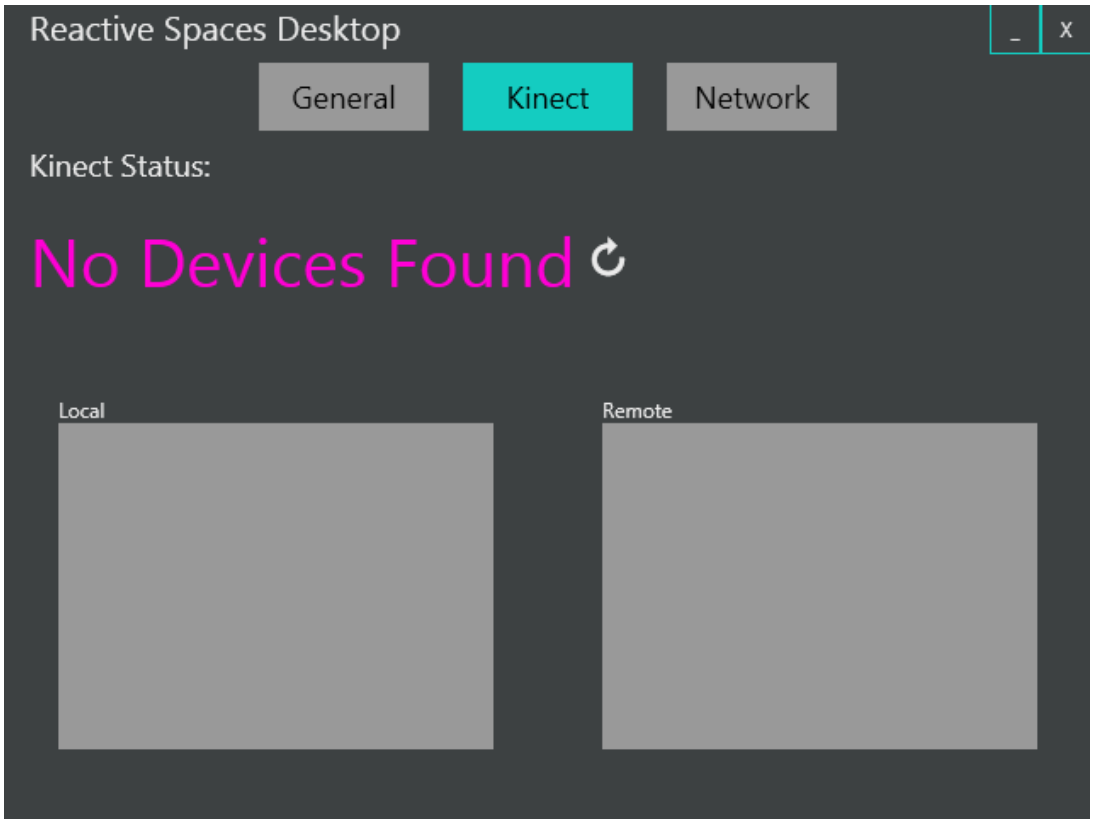
LAYOUT : GENERAL

This area contains information about the overall state of the app as well as information about the connected game instance and local websocket.



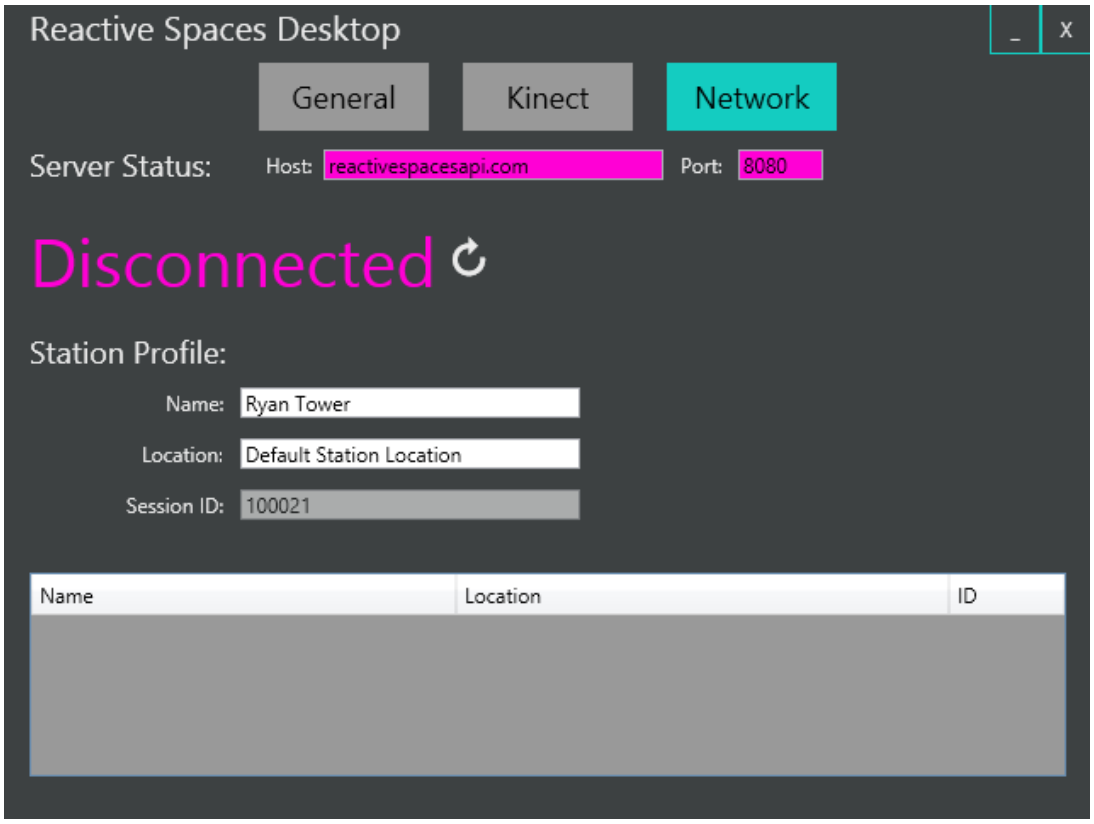
LAYOUT : KINECT

This area will display skeleton information for local and remote Kinect as well as an overall status of the Kinect system.



LAYOUT : NETWORK

This area contains information relating to the server connection. This is where users will enter information about their station. It will also provide and overall connection status as well as a list of connected peers.



VISUAL STYLE : API

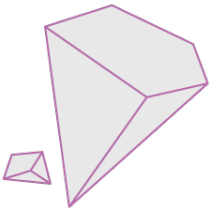
The desktop application will have a clean and memorable design. The colours shown in colour scheme below will be the main colours used throughout the application as well as the website, promotional materials, and the example game. The colours will consist of mainly shades of grey with the accent colours being shades of cyan and magenta.

#E7E8E8	#979898	#3D4142	#60C4B9	#B76BAB

VISUAL STYLE : LOGO

Below is the logo that will be used in the application. It will be used to create a brand out of the API.

MINIMAL :



MONOCHROME :

COLOURED :

2.2.2 SERVER PLATFORM

- JavaScript code running on Node.js server
  - Linux server
  - Node.js with Forever
  - Publicly accessible internet connection

FUNCTIONALITY

- Accept and manage connections from client apps
- Match clients into sessions based on connected games/apps
- Pass messages between clients in the same session
- Notify clients of changes in their session

MESSAGE FORMAT

- All messages in the system will take the form of JSON objects
- Each json object is to be comprised of:
  - Type: an integer representing a message type enumberator
  - Message: the string of another JSON object carrying the specified type of data
- The message string's content will vary based on the message, and could also be any Javascript object sent by the user
- Some message strings may not be decoded by the server, but only the JavaScript library and/or the Desktop app.

EXPOSED FUNCTIONALITY

A. MAIN FUNCTIONS

- Web browser with web socket support
  - Internet Explorer 10 or greater
  - Firefox 31 or greater
  - Chrome 31 or greater
  - Safari 7 or greater
  - Opera 24 or greater

B. EXPOSED FUNCTIONALITY

- See Appendix A for a complete breakdow

3 DEMO GAME

3.1 GOALS

- Showcase the API that is created.
- Illustrate API features and functionality that will help game developers
- Show off how to use the API

3.2 MECHANICS

PRIMARY FORM CIRCLES :

The player's goal is to collect a certain amount of the form circles which then turns into a larger circle.

LARGE CIRCLES :

The larger circle collects the score triangle items to increase players score. When one station successfully creates a larger circle, it appears on all active stations. This larger circle helps the creator collect score not only on their station, but on all other stations. This large circle has a lifespan and after it is complete it will fade away. This allows for the need to keep creating large circles.

GESTURES :

The stations can also affect each others environment, making it harder for the other stations to play. This is done through the movements of the players', each affecting the other's environment.

ROUNDS :

When the game is started it starts a new round that any one can connect to. The rounds last 1 minute. After the round is complete it displays the current stations score, a list of all other stations' scores, and a countdown timer until the next round begins.

3.3 GAMEPLAY

PLAYERS :

The game is played between two connected computers. The players' main goal is to collect the highest amount of score possible.

GESTURES :

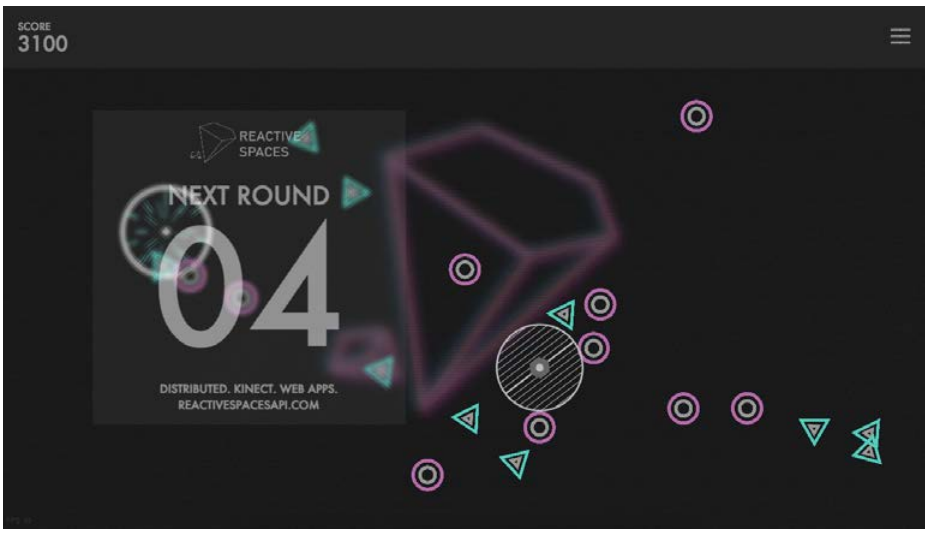
This is done by first collecting primary form circles to form a larger circle, the player does this using their hand positions that are mapped on to the screen and displayed using the hand icon.

SCORING :

The large circle then moves around the screen in a random pattern on its own collecting the score triangles, which adds points to the station's score.

3.4 INTERFACE

The game will have a simple interface for players. The interface will display the players score along the top of the screen. The menu for the game can be accessed by the player holding their hand in the top right corner. This will display all connected players scores.



3.5 ART STYLE

The game is going to represent the API's functionality so the art style is going to have the same branding elements as the API. The game will follow the colour scheme of the API using mainly different shades of grey with the accent colours different for each station. The style will also reflect the style of the gameplay is quite simple and abstract. The overall look of the game is a simplified outer space theme. The background will be solid black to allow the game elements to pop.

FORM CIRCLES :

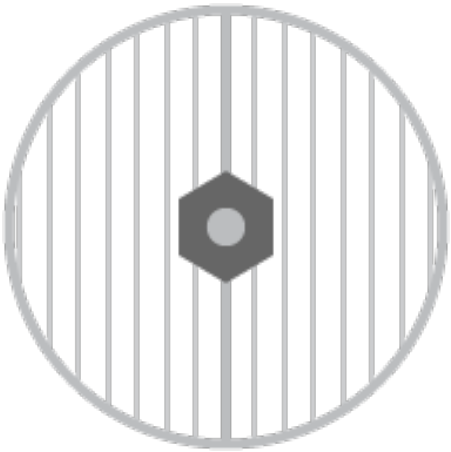
The form circles will consist of two rings, one inner white ring and an outer coloured ring.

SCORE TRIANGLES :

The score items contrast the form circles in shape and colour. They will be triangular and a contrasting colour of the form circles.

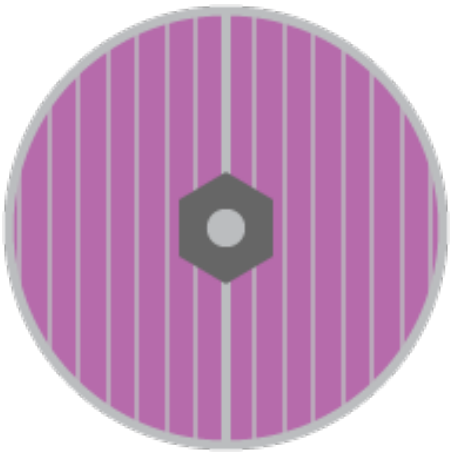
HANDS :

The hand symbols consist of a sphere and a hexagon around it. There will be two hand symbols per player.



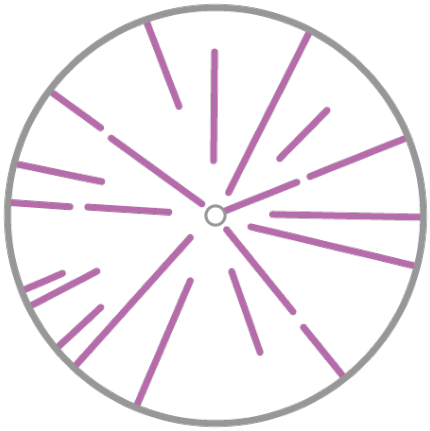
FILLING HANDS :

As the hand collects the form circles, the hand symbol will fill to create the large circles.



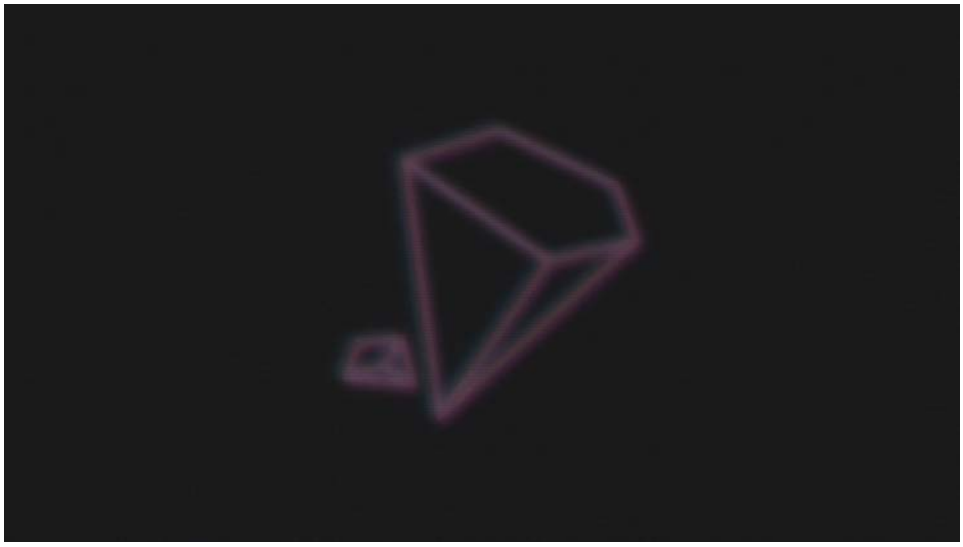
LARGE CIRCLES :

When the filling hands become full, they create a new large circle. Large circles also appear if they are created in another station's game, and these are differentiated by the color of the circle.



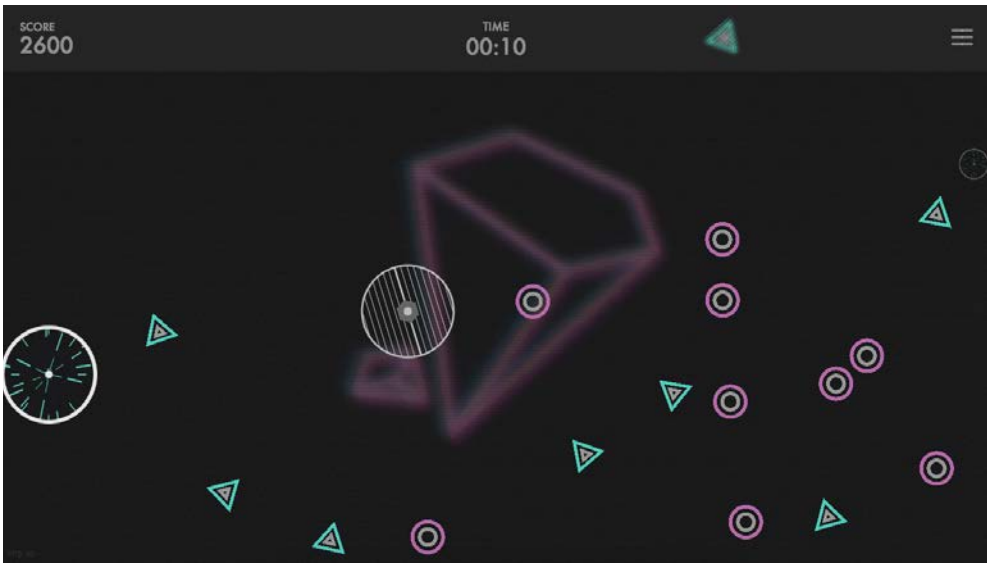
BACKGROUND :

The background was created at add more depth to the demo game.

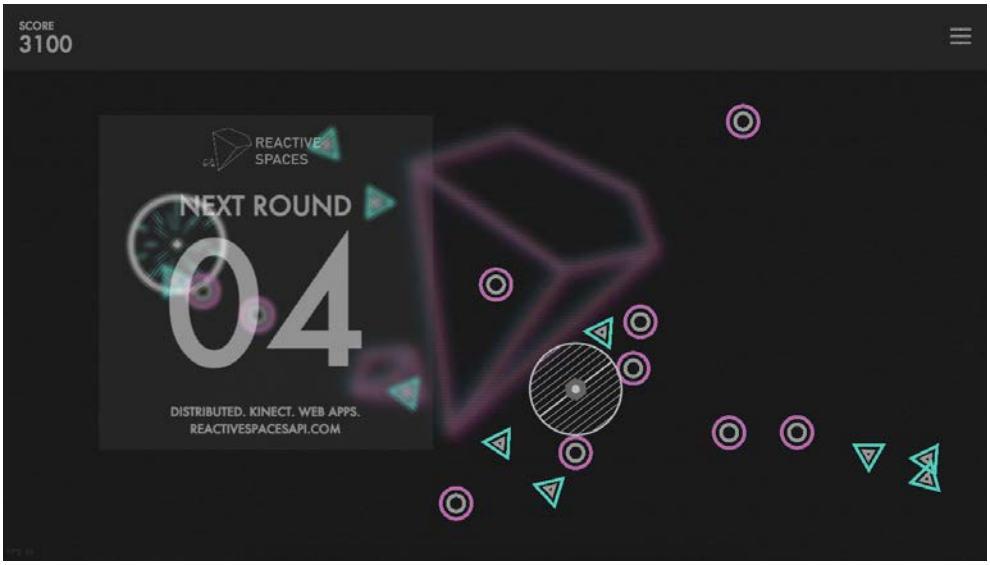


3.6 CONCEPTS

GAME :



MENU :

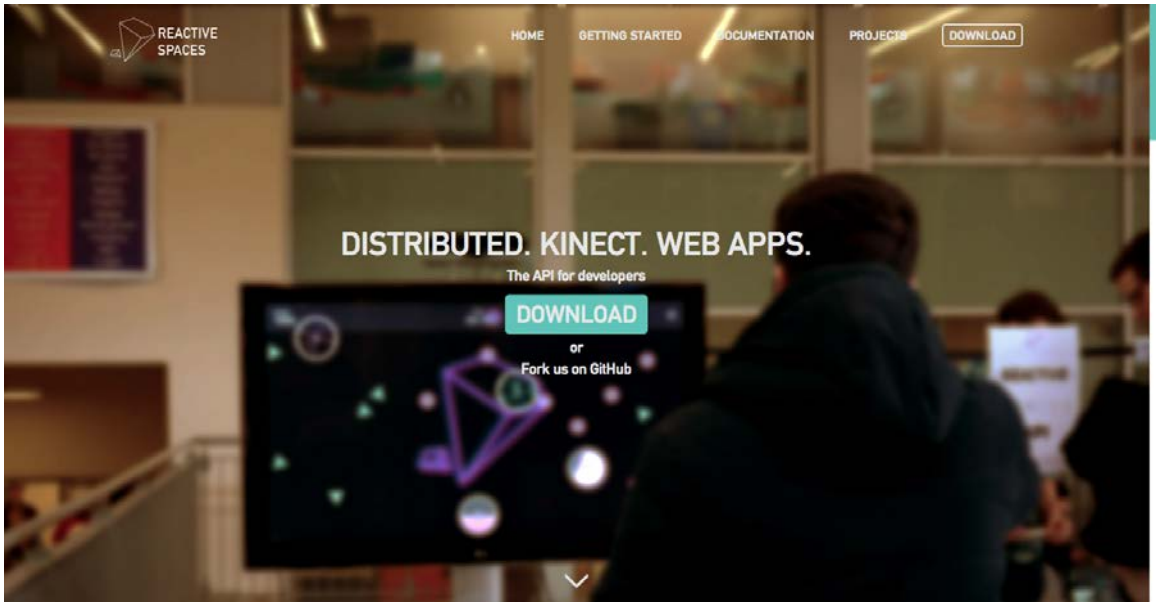


4 PROMOTIONAL PLAN

4.1 WEBSITE

The website will contain information about the API and example games developed with the API. There will be a download link for the API available on the website for developers to start developing with. The website will list the key features of the API, the documentation for developers, and contact information. The website has a colour scheme that will be incorporated in each of the promotion materials and match the design of the desktop API and the example game. The website will be the main medium for promotion.

ReactiveSpacesAPI.com



4.2 POSTERS / STICKERS

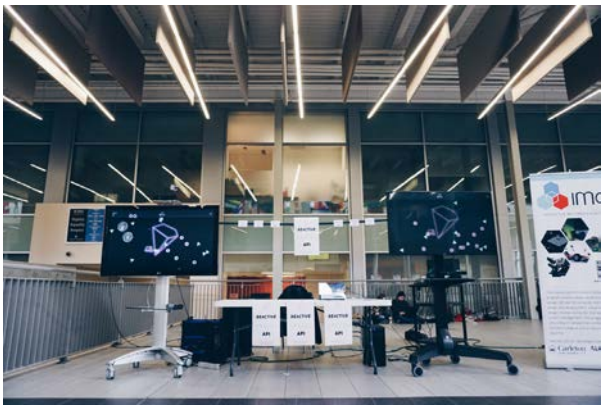
The posters and stickers will be used to create interest for programmers and to get them to visit the Reactive Spaces API website. From the team's experience, programmers love stickers. On the demonstration day the team will be handing out stickers for interested parties to keep. The posters will be used both electronically and physically to promote the API and live demonstration day.

POSTER

STICKER

4.3 LIVE DEMONSTRATION / DEMONSTRATION VIDEO

The demonstration day will be held on Monday December 1st and will be used to show off the API with the use of the example game. The demonstration will be stationed at the fourth floor university center from nine in the morning to three in the afternoon. There will be two televisions each running the game with kinect to show off the features of the API. During the live demonstration the team will be giving out info cards and stickers to whomever would like them. The demonstration will also be filmed in this time period to promote the API on the website.



DEMO VIDEO :

https://www.youtube.com/watch?v=5822Q7wYUeo

5 LIST OF ABBREVIATIONS

- JS - Javascript
- p2p - peer to peer

6 APPENDIX A - JAVASCRIPT LIBRARY MEMBERS AND METHODS

All data and functions exist in a global object accessible as *ReactiveSpaces* or simple RS.

6.1 SETTINGS

RS.BASEURL :

Stores the base host/url for connecting to the desktop client. In most cases, this should not be changed. If you wanted to try launching the desktop app on another PC and then use its Kinect, it should theoretically work by changing this address.
Default value : "ws://localhost"

RS.LOCALPORT :

Stores the port on which to connect to the desktop app. Only change this value if you needed to change the connection port in the desktop app.
Default value : 8080

RS.MESSAGE_DELAY :

Stores the number of milliseconds delay required between each sent message. Shortening this value could cause instability and lag in the system and server. If you're game requires messages more often than this, you should look at using a true multiplayer system.
Default value : 100

RS.SUPORTED_PLAYERS :

Represents the maximum number of local players that will ever be tracked. Changing this value has no effect.
Default value : 2

RS.MOVEMENT_BLENDING :

Boolean value denoting weather or not to blend skeleton frame data and create smoother joint movement. Turning this off can improve performance, but blended joint positions will no longer update.
Default value : true

6.2 MEMBERS

RS.socketSupported :

Stores weather or not the browser supports web sockets. If false, Reactive Spaces will not be able to connect to the desktop app or server.

RS.socket :

Stores the current socket connection to the desktop app (if any).

RS.connected :

Boolean value representing the current connection status of Reactive Spaces.

RS.lastMessage :

UNIX timestamp denoting the last time a message was sent from Reactive Spaces to the desktop app.

RS.station :

Stores the *RS.StationProfile* object representing the local station information. If no connection to the desktop app has been made, this value can also be null.

RS.remoteStations :

Stores a list of *RS.StationProfile* objects representing the remote stations that are in the same game session.

RS.players :

Stores a list of *RS.Skeleton* objects representing the current local players.
RS.remotePlayers
Stores a list of *RS.Skeleton* objects representing the current remote players.

RS.lastBlendUpdate

UNIX timestamp denoting the last time the blended skeleton data was updated

RS.listeners :

Stores all event listeners added to Reactive Spaces

RS.messenger :

Provides basic error logging capabilities for Reactive Spaces.
Also defines a Global Message object which has an enumerator for message types defined as
Message.type = {"MESSAGE" : 0, "WARNING" : 1, "ERROR" : 2}

RS.appInfo :

Stores information about the connected web app. Members are: name, version, maxPeers.
Default Value:{
name: "Default App Name",
version: 0,
maxPeers: 4
}

6.3 METHODS

RS.Connect (appName, appVersion, features, port)
appName : name of web app that is connecting
appVersion : version of app that is connecting (int or float). Only app instances with the same version will be matched on the server
features : list of features that are required by the app. Use enumerator RS.Features.
port (optional): connection port to use. If omitted, RS.LOCALPORT will be used
Attempts to connect to the Reactive Spaces Desktop App. Returns true or false depending on whether connection was successful or not.

RS.Disconnect ()
Disconnects from the desktop app and server.

RS.BlendUpdate ()
Updates the blended skeleton positions. Called automatically when *RS.MOVEMENT_BLENDING* is true on connection. Can be called manually on your apps update loop if desired.

RS.ActivateMessenger ()
Reactive Spaces provides an optional messenger system to display messages, warnings and errors easily.
Include the messenger in you project with
<script src=" http://reactivespacesapi.com/lib/Messenger.js"></script>

And then call this function to upgrade RS.messenger to this advanced messenger. Post your own messages with RS.messenger.display(Messenge. type type, string message, string solution).

RS.addEventListener (event, callback)
event : string denoting the envent to listen to. Use enumerator RS.Events or see RS.Events in the documentation for a full list
callback : function pointer for function to be called when event fires

RS.removeEventListener (event, callback)
This function currently has no effect

RS.Send (object)
object: javascript object to send
Sends the given object to all stations in the current app session.

6.4 TYPES

RS.Skeleton ()
Describes the skeleton object that ReactiveSpaces uses to store Kinect data

RS.Skeleton.Update (skeleton)
skeleton: the skeleton object to copy the data from
Updates the joint positions and data from another skeleton object. This is used for incoming data streams from the desktop.

RS.Skeleton.BlendUpdate (deltaTimeS)
Blends previous skeleton with current skeleton. ****

RS.SkeletonJoint (skeleton)
skeleton: the skeleton object to copy the data from
Describes a skeleton joint object

RS.SkeletonJoint.SetFromJoint (joint)
joint: valid RS.SkeletonJoint object
Copies data of the given joint into this one

RS.SkeletonJoint.BlendUpdate (deltaTimeS)
Called by ReactiveSpaces to update the smoothed screenPosition. Also updates joint velocities.

RS.Vector3 (x, y, z)
x (optional): inital x value, default is 0
y (optional): inital y value, default is 0
z (optional): inital z value, default is 0
A simple object which defines a 3D point (x, y, z)

RS.Vector3.Set (x, y, z)
x (optional): inital x value, default is 0
y (optional): inital y value, default is 0
z (optional): inital z value, default is 0
Sets the value of this vector (x, y, z)

RS.Vector3.SetFromVector (vector)
vector: object to copy from {x, y, z}
Sets the value of this vector from another vector. If x, y, or z are unset on the object it sets 0.

RS.Vector3.AddVector (vector)
vector: object to add {x, y, z}
Adds given vector.

RS.Vector3.SubVector (vector)
vector: object to subract {x, y, z}
Subtracts given vector.

RS.Vector3.MultiplyScalar (scalar)
scalar: number to multiple vector by
Multiplies the vector by given scalar.

RS.Vector3.Normalize ()
Normalizes vector, length of vector = 1.

RS.Vector3.Length ()
Returns the length of the vector.

RS.Vector3.LengthSqd ()
Returns the squared length of the vector.

6.5 UTILITY FUNCTIONS

RS.DrawSkeleton (context, skeleton, color, blend)
Draws given skeleton on the given 2D context. Maps screen position to the bounds of the canvas.
context: valid 2D drawing context for an html 5 canvas
skeleton: valid skeleton object from ReactiveSpaces
color (optional): string canvas color identifier, default is "#FFF"
blend (optional): boolean - sets weather or not to use the blended skeleton. RS.MOVEMENT_BLEND must be set to true (default is false)

6.6 ENUMERATORS

RS.Features

Shows what features are supported.

Kinect	0

RS.JointTypes

To pick joints from the skeleton object.

HEAD	0
SHOULDER_CENTER	1
SPINE	2
HIP_CENTER	3
SHOULDER_LEFT	4
SHOULDER_RIGHT	5
ELBOW_LEFT	6
ELBOW_RIGHT	7
WRIST_LEFT	8
WRIST_RIGHT	9
HAND_LEFT	10
HAND_RIGHT	11
HIP_LEFT	12
HIP_RIGHT	13
KNEE_LEFT	14
KNEE_RIGHT	15
ANKLE_LEFT	16
ANKLE_RIGHT	17
FOOT_LEFT	18
FOOT_RIGHT	19

RS.MessageTypes

Enumerator for incoming web socket messages.

APP_INFO	0
CUSTOM	1
KINECT	2
REMOTE_KINECT	3
STATION_PROFILE	4
PEER_CONNECT	5
PEER_UPDATE	6
PEER_DISCONNECT	7
LOCAL_PLAYER_ENTER	8
LOCAL_PLAYER_EXIT	9
REMOTE_PLAYER_ENTER	10
REMOTE_PLAYER_EXIT	11
FEATURE_MISSING	12

RS.Events

A simple list of event types dispatched by ReactiveSpaces.

connect
disconnect
localkinect
remotekinect
message
featuremissing
stationlocal
stationconnect
stationupdate
satationdisconnect
localplayerenter
localplayerexit