School of Electronics and Computer Science

Faculty of Physical and Applied Sciences

University of Southampton

Date: January 15, 2012

COMP3001 Group Scripting Coursework - Team B

Team Members:

Dexter Lowe (dl10g09@ecs.soton.ac.uk) James Skinner (js40g09@ecs.soton.ac.uk) Adam Thomas (ajt1g09@ecs.soton.ac.uk) Bahman Asadi (ba2g09@ecs.soton.ac.uk) Peter West (pw9g09@ecs.soton.ac.uk) Alex Parker (ajp3g08@ecs.soton.ac.uk) Tom Smith (taes1g09@ecs.soton.ac.uk)

http://3001Blokus.appspot.com

1 Description of Prototype Functionality

1.1 Blokus

Blokus is a 2 or 4 player strategic board game, where each player takes turns in placing coloured polyominoes (footnote: A polyomino is a 2D geometric shape formed of one or more equal squares edge-to-edge) on a 20x20 board with the aim of placing all 21 of their pieces. For a 4 player game, each player controls one colour, and for a 2 player game, each player controls two colours. Each players first turn must involve placing one of their pieces on one of the four corners of the board. Each subsequent turn must involve placing a piece with only corners touching their existing pieces (so there may be no flat edges of the same coloured shapes touching). No piece may overlap any other piece. Players may employ tactics to block another player from placing their pieces. When no further moves are possible by any player the game is over. A colour loses a point for each square in each unplaced piece. They gain 15 points for placing all their pieces and a further five if the final piece they placed was their monomino (single square).

Client-side validation is used to highlight the area the piece would cover to show whether a position is valid or invalid as appropriate. If the user wishes to rotate or flip the a piece they either hover over it in their piece-tray and use the control halo that appears or once picked up they can use left and right to rotate or h and v to flip horizontally and vertically respectively. Once the user has decided where to place it they click once more to place the piece on the board.

1.2 Authentication

Users are able to play games without logging in, however, those who register will be able to track their statistics about games they have won and lost. Three methods are available: registering with an account that will be native to the Blokus application, logging in with Facebook, or logging in with Google.

1.3 Lobby Screen

From the initial screen that this user is shown - the lobby screen - the user may play a 2 or 4 player game, quick game. Selecting one of these will place the user into a waiting room until there are enough players to start a game, where they will be taken to the game screen. Additionally, the user may select Private Game, where they will be given a unique url to share with their friends, so that they can play a game with only people that they know. While waiting for these players, they will be placed in the waiting room.

2 List of Tools and Techniques Used

2.1 Development Tools

- Various Text Editors with syntax highlighting for general development. No language specific IDEs were used to reduce platform incompatibility issues.
  - Sublime Text 2 (All)
  - Vim (Unix Based Environments)
- Google Chrome Developer Tools and Firebug were used for Javascript debugging.
- Mercurial (hosted on BitBucket) was used for collaborative source control. Mercurial was chosen over SVN due to its superior branching and its local repository.
  - Mercurial (Command Line) (All)
  - TortoiseHG (Windows Environments)
- Mozilla Firefox, Google Chrome, Chromium, IE9 and Safari were used for testing.
- mjson.tool was used to check the information in python objects.
- gvimdiff, Tortoise Diff, WinMerge, and KDiff3 were used to solve merge conflicts.
- cron was used to garbage collect completed or abandoned games and inactive temporary users from the database.

2.2 Project Management

- Issue tracking was done using BitBucket
- Google Docs were used for collaborative authoring and document sharing.
- Facebook was used for group contact, communication and organisation.

2.3 Techniques

We used a RAD (Rapid Application Development) approach to development having regular meetings and aiming to have a prototype with more functionality at each one. We also used Facebook and Google Docs to ensure that everyone knew the current state and design of the system.

Temporary data was created on the client side and a DEBUG page to emulate many server functions so that front-end development could continue even if the back-end was not working. In this way the various modules of the system were kept separate as far as possible. Also we used AJAX and JSON based techniques to simplify dynamic web pages and data transfer between the front and back end.

3 Relevant Statistics

3.1 External Sources

3.1.1 CSS

css-reset Resets all browser layout so all browsers are more likely to be the same.

input A small script used to place labels inside input text fields.

3.1.2 Python

django-nonrel Lets Django work with non relational databases like on the GAE.

tastypie Creates REST interfaces for Django web-services.

django-guest Allows guest accounts and handles garbage collection for them.

social_auth Allows authenticating users via Facebook and Google, among others.

3.1.3 Javascript

labels Places labels for controls inside the control as seen on the login screen.

jquery Allows easier manipulation of the DOM.

underscore Provides powerful tools to work with advanced data types.

backbone Provides simple routing and provides client side representation of models.

Raphaël Provides a simple and powerful interface for SVG.

3.2 Relevant Statistics

Language	Files	Comments	Code	Purpose
Javascript	13	101	1682	GUI, navigation, validation and back-end interaction.
Python	14	127	777	Server side scripting.
CSS	1	4	363	Client side styling settings.
HTML	6	4	325	Templates and static information.
YAML	2	0	36	Google App Engine instructions.

4 Design and Implementation

4.1 Design

The design goals for the user interface were to emulate the Blokus board game as far as possible and make it simple to jump into a game and start playing. To achieve this we added the ability to play the game as a guest without registering, and we also provided Quick Play and matchmaking modes that automatically select your opponent along with the private lobby where you can invite your friends to play. If the user wishes to save their scores we added the option to register a profile and sign in. To reduce the burden of registering a new profile and password we added the option to sign in via a Google account or a Facebook account.

Market research conducted on existing Blokus games discovered that the interface provided to rotate and flip the pieces was inefficient, requiring 4 or more clicks to use. Our design streamlines the process by displaying the rotate and flip buttons around the piece when the mouse hovers over a piece, reducing the total number of clicks to 2 or more. Finally piece validation is run continuously when a piece is selected indicating to the user if the current position is valid or not; an improvement over existing implementations.

4.2 Implementation

The Python Django framework was selected for the server due to existing knowledge of the technology within the team, helping simplify the server development. The many Django packages available also streamlined features such as guest login and social authorization.

A restful interface was implemented to handle the communication between the client and the server. This also forced us to define a clear REST API specification enabling us to break down the work cleanly between group members and provide flexibility for the client interface. The Django models and tastypie extension were used to automatically generate both the REST API and database structure from the same file.

HTML 5, SVG and CSS 3 in combination with Javascript are used to provide a modern user interface to the Blokus game. Through the use of the JQuery and backbone libraries all communication with the server (excluding the registration and social authorization actions) are handled by AJAX calls to the servers REST API communicating via JSON objects using HTTP GET and POST. The backbone library also provided a model of the servers REST interface and navigation features permitting the Javascript to be split into multiple views.

The game board and pieces were rendered with the Raphael SVG library that permitted us to abstract the game board drawing into simple Javascript calls. In addition piece placement was validated on the client side in Javascript improving usability.

5 Evaluation

When considered as a whole this project has been a success both educationally and productively. It is possible using our system to quickly and easily find and play a game of Blokus. The actual game-play has also been fully implemented including a very responsive and intuitive interface. Further more we have achieved many of our secondary objectives such as a simple and self explanatory lobby allowing 4 player, 2 player and private games. We have also successfully integrated with both Google and Facebook for social authentication but made sure to provide our own authentication system for those who either dont have or dont want to share their information from these external sites. We also managed to keep to our rapid-play objective throughout all areas of the game and thus we managed to integrate guest user accounts into our system such that just for a quick game players do not even need to register.

Despite the large range of features implemented other features were considered but not implemented due to the time constraints. We initially considered having the system post to the social networks of our users, device specific controls and, more ambitiously, to create AI players. However it was decided to focus on the aspects of the system more intrinsic to correct operation so these were not implemented.

The Python back-end successfully keeps track of both the game states of those games currently in progress and the user information. The amount of data transmitted is minimised by not creating a game until there are enough players to start the game and only sending the changes in game state rather than the entire game each time. This was an important point to achieve as we wanted the game to feel very responsive. This desire for responsiveness also led to us performing validation on the client side so that the validity of a move can be seen real-time without the need to submit to the server. As no input from the client should be trusted, validation is also implemented server-side, also, after each move Pyhton works out if the game is over and calculates the scores. This is again because anything sent by the client cannot be trusted.

The Javascript side of the system has also been a success. It provides an interactive, intuitive and responsive interface and periodically polls the server to collect any changes to the game state. We also have it performing validation on the fly in order to make it clear to novice users whether a position is valid or not. This is seamlessly integrated into the game flow and does not require the user to do anything extra to gain this ability once more holding to our rapid-play aim.

Overall this project clearly demonstrates our ability to use Javascript to provide responsive and intuitive interfaces with and tie them asynchronously to a back end application written in Python using a Rest API. Both sections of the system demonstrate our ability to model an existing real world system and adapt it for other purposes. Our lobby system shows our ability to handle many discrete requests and appropriately process them. While the system does have areas where it could be extended, it is a fully-functional prototype, and more time and resources the game and surrounding functionality could be further improved. In conclusion we believe that this prototype demonstrates our ability to use the power of scripting languages to combine many components to create a well structured, stable and cohesive system and it has thus achieved its goal.