Gameblox Flexidor: Adding Flexibility to Blocks Based Programming Environments
by
Erica Du
S.B., Massachusetts Institute of Technology (2014)
Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science
at the
Massachusetts Institute of Technology
June 2015

© Massachusetts Institute of Technology 2015. All rights reserved

Signature redacted

Department of Electrical Engineering and Computer Science
May 23, 2015

Signature redacted

Professor Eric Klopfer
Director, MIT Scheller Teacher Education Program
Thesis Supervisor

Signature redacted

Professor Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

1
Abstract

Gameblox is an online, graphical programming environment currently being developed by the MIT Scheller Teacher Education Program as a means of making game development more accessible to students with little to no programming background. Through direct manipulation, users can create complex games by simply dragging objects onto specified screens, and attaching blocks as executable instructions. Users can create, modify, and test their games, all in one environment. However, certain users, such as beginners or students, may find that a scoped editor with only a subset of Gameblox features would be easier to use and still fulfill their goals. This thesis outlines the design and implementation of a new tool within Gameblox called the Flexidor Builder, which will allow any user to selectively restrict the view of the full Gameblox editor to create custom Gameblox editors with only a subset of features. This restricted editor is called a flexidor.
Acknowledgements

First, I would like to thank Eric Klopfer for inspiring my interest in educational games, and for giving me the opportunity to work on Gameblox, as well as a slew of other cool projects in the STEP Lab throughout my time at MIT. Working on this project in particular has been an extremely valuable experience, and could not have been possible without his intellectual support and guidance.

Second, I would like to extend a huge thank you to Paul Medlock-Walton for his tireless dedication to Gameblox, and steadfast support of my thesis project throughout this year. His mentorship and guidance were invaluable, and his role, irreplaceable. His help during the design, coding, and evaluation periods were vital in the completion of this thesis, and I have learned so much from him. His ability to balance his roles continually impresses me, and I am grateful to have had the opportunity to work alongside him.

Third, I would like to thank the STEP Lab for being a constant source of encouragement, fuel, feedback, and ideas. Passion and hard work are contagious, and I could not have asked for a better environment to work in this past year.

Lastly, I would like to thank my friends and family for their help, love, and support. Without them, I would not be who or where I am today.
Contents

1 Introduction 14
   1.1. Motivation for Flexidors ................................................................. 14
      1.1.1. Overview .................................................................. 14
      1.1.2. Low Floors, High Ceilings ........................................... 15
      1.1.3. Zone of Proximal Development .................................. 15
   1.2. Motivation for Flexidor Builder .................................................. 16
      1.2.1. Instructional Scaffolding .............................................. 16
      1.2.2. Use Cases .................................................................. 16

2 Background 18
   2.1. Existing Simplifications .............................................................. 18
      2.1.1. Blocks Based Programming Languages ......................... 18
      2.1.2. Multi-setting Interfaces ................................................... 19
      2.1.3. Templated Games .......................................................... 20
      2.1.4. Restricted View .............................................................. 20
   2.2. Gameblox .............................................................................. 21
      2.2.1. Overview .................................................................. 21
      2.2.2. Design ..................................................................... 22
      2.2.3. Blocks ..................................................................... 23
      2.2.4. Play ....................................................................... 26

3 Flexidor 28
   3.1. Overview ............................................................................. 28
      3.1.1. Introduction ................................................................. 28
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Dialog box to select level of expertise in DrScheme interface</td>
<td>19</td>
</tr>
<tr>
<td>2-2</td>
<td>Gameblox Design Panel</td>
<td>23</td>
</tr>
<tr>
<td>2-3</td>
<td>Gameblox Blocks Panel</td>
<td>25</td>
</tr>
<tr>
<td>2-4</td>
<td>Gameblox Blocks Panel (Events Drawer Open)</td>
<td>25</td>
</tr>
<tr>
<td>2-5</td>
<td>Gameblox Play Panel</td>
<td>26</td>
</tr>
<tr>
<td>3-1</td>
<td>Example Flexidor Design Panel</td>
<td>32</td>
</tr>
<tr>
<td>3-2</td>
<td>Example Flexidor Design Panel</td>
<td>33</td>
</tr>
<tr>
<td>3-3</td>
<td>Example Flexidor Blocks Panel (drawers closed)</td>
<td>37</td>
</tr>
<tr>
<td>3-4</td>
<td>Example Flexidor Blocks Panel (open drawer)</td>
<td>37</td>
</tr>
<tr>
<td>3-5</td>
<td>Example Flexidor Blocks Panel (open drawer)</td>
<td>38</td>
</tr>
<tr>
<td>4-1</td>
<td>Flexidor Builder Design Panel Form Mockup, with original name “Flexditor”</td>
<td>42</td>
</tr>
<tr>
<td>4-2</td>
<td>Flexidor Builder Design Panel Visual Overlay Mockup, with original name “Flexditor”</td>
<td>42</td>
</tr>
<tr>
<td>4-3</td>
<td>Flexidor Builder Home Screen</td>
<td>44</td>
</tr>
<tr>
<td>4-4</td>
<td>Flexidor Builder Design Panel Form (Final Design)</td>
<td>46</td>
</tr>
<tr>
<td>4-5</td>
<td>Flexidor Builder Blocks Panel Form</td>
<td>47</td>
</tr>
<tr>
<td>4-6</td>
<td>Flexidor Builder Example Blocks Panel Form Use Case</td>
<td>47</td>
</tr>
</tbody>
</table>
List of Tables

5-1 Treatment for Flexidor Study ................................................................. 52
5-2 Data Collected for Flexidor Study ......................................................... 54
5-3 Answers to Pre-Questionnaire ............................................................... 56
5-4 Average Self Reported Difficulty ......................................................... 56
5-5 Post-Questionnaire Data ................................................................. 56
5-6 Average Self Reported Comparisons ................................................. 57
5-7 Error Descriptions and Frequencies .................................................... 57
5-8 Data Types for Flexidor Builder Study ................................................ 64
5-9 Ease of User Responses ................................................................. 64
5-10 Learnability Responses ................................................................. 65
5-11 Completion Statistics ................................................................. 65
Chapter 1

Introduction

1.1 Motivation for Flexidors

1.1.1 Overview

Game development, like software development, has generally been considered inaccessible to the average, non-programming population due to its reliance on technical knowledge [1]. In recent years, tools have been built to reduce the technical barrier of learning how to code [2]. Tools have also been created to empower people who have very little technical background to develop their own games. Gameblox, an online, visual-based game development environment, is one such tool.

Gameblox is powerful, and provides the user with many options for manipulation. However, if a user has a specific purpose or project in mind, he or she may only employ a small subset of all the features in Gameblox, and find the rest distracting. These scenarios motivate the existence of Gameblox editors with limited functionality, as they create simpler, more specific, analogous interfaces. This new, restricted Gameblox editor will be called a flexidor, as a concatenation of “flexible editor.”
1.1.2 Low Floors, High Ceilings

“Low floors, High Ceilings” refers to the principle of making the onboarding process easy for beginning learners while providing expert users sufficient tools and variants to create more sophisticated projects. This design philosophy was the main motivation behind the creation of Scratch, one of the most popular blocks based programming languages in use today. This principle is also the motivating principle behind flexidor creation. Having a low floor means giving beginners the immediate confidence to succeed and keep learning, through an interface that is not intimidating [3].

Whereas experts would view the breadth of options in the full Gameblox editor as opportunities to use their creativity, the same number of options may overwhelm and flood a novice game designer, who may pick simplicity over freedom when still trying to learn the basics. Currently, Gameblox provides a low floor for beginners through a colorful, highly visual interface, as opposed to a text-heavy editor. However, there is not much of an onboarding experience, and users are presented with all the features immediately, with no direction on where to first focus their attention. Beginners may need to start with something that has a smaller scope, with limited functionality and options.

1.1.3 Zone of Proximal Development

An idea proposed by Lev Vygotsky, the zone of proximal development (ZPD) describes the region between a student’s individual abilities and the level of problem solving that the particular student can attain through collaborating with or receiving help from more capable peers or an instructor [4]. It then follows that learning can be facilitated by the use of a temporary support structure until the student can achieve the same results on his or her own. The same concept can be applied to a student’s ability to create certain types of games using Gameblox. A flexidor may be able to serve as the temporary support structure students use during this intermediary step.
1.2 Motivation for Flexidor Builder

1.2.1 Instructional Scaffolding

Instructional Scaffolding describes the temporary support structures, both either social or cognitive, instructors provide to help their students reach new understandings they could not reach on their own. Instructors need a tool to create these structures, as each class’ technical abilities and goals are different [5]. The Flexidor Builder would be one such tool.

1.2.2 Use Cases

eduX Courses

Gameblox was the main online game development tool used in two eduX courses offered by the STEP Lab, 11.127x (Computer Games and Simulations for Investigation and Education) and 11.126x (Introduction to Game Design and Development). Since the two courses were project based, throughout the two courses, students used Gameblox to create games to apply the game design lessons learned. Course staff could create a series of flexidors to teach one specific game design concept at a time, by targeting different areas of the Gameblox editor.

Tutorials

Flexidors could also be used in tutorials to teach users how to interact with the full Gameblox editor. Because flexidors react to user interactions in the same way a full Gameblox editor would, it could potentially be used as a tool to gradually teach users specific aspects of Gameblox behavior. The Gameblox team, or course staff in a class using Gameblox, could build these custom flexidors to accompany tutorials using the Flexidor Builder.
Chapter 2

Background

2.1 Existing Simplifications

2.1.1 Blocks Based Programming Languages

Many game development environments have been built to abstract the programming logic component, employing a more graphics-oriented, user-friendly interface for users to build the game logic. Scratch, developed by Mitchel Resnick’s Lifelong Kindergarten Lab, is one such tool [6]. However, Scratch is not an application exclusive to game creation, and can be used to make anything from animations to stories.

Graphical programming languages are especially appealing to children. Though kids may be intimidated by the syntax of programming languages, they may be familiar with the concept of “blocks,” due to the analogy with a popular physical toy. Other advantages of visual programming languages over text based ones include more visual cues (e.g. designing blocks to fit together only in a specific way), higher glanceability (e.g. ability to determine functionality of program faster), and a visualization of the pathway taken by code [4].
2.1.2 Multi-setting Interfaces

Interfaces providing multiple settings, often two, also decrease complexity. These tools usually include a toggle between a beginner and advanced setting, indicating the breadth of features a user wants to view. For example, GameMaker Studio, a game development tool, caters to novices and experienced professionals by providing two such settings. There is also a free version with a limited set of support and features, and a paid version with a broader set of capabilities.

Dr. Scheme, a visual tool developed by researchers at Rice University to teach programming concepts through the Scheme language, has a more fine-grained setting system. Dr. Scheme stratifies users into four different levels of ascending difficulty in order to protect students from using language features that have not yet been introduced in class. Each language level builds upon the previous, from beginner all the way to the full scheme language, as each level has its own discrete set of new features. This separation of language features allows for more controlled learning, as well as easier debugging, both of which are particularly useful for learners [5].

Figure 2-1: Dialog box to select level of expertise in DrScheme interface.
2.1.3 Templated Games

Some game development tools eliminate programming altogether. These cater to a casual interest in game creation by providing all the necessary game logic, shifting the focus from the creation of a unique game, to the creation of a custom version of their favorite, existing games. Many of these tools have templates that allow users to upload their own assets, and manipulate object positioning within the game, allowing a user to create their own games with very little effort and zero technical knowledge.

Another way that templates manifest themselves in these visual programming environments are through extensions of a set of complete projects. StarLogo Classic, a programmable modeling environment created at MIT, employed a Template Wizard to allow users to create different types of turtles, while offering a set of pre-defined behaviors to choose from. Users could define different “breeds” by creating a name, and then manipulating a slider interface to assign different variable values for the pre-set functions. Users could also extend the template by creating custom functions for their models. This wizard was meant to get beginner users started on a project, and then to extend it using the full programming interface after obtaining some degree of mastery [6]. NetLogo, a multi-agent programming environment created by Northwestern University, has a library of model games and applications that users can tweak and extend. For example, there is a Pac-Man model game that is included in the NetLogo application download. Here, users are encouraged to extend the game with their own power-ups, bonuses, and enemies [7].

2.1.4 Restricted View

Simpler environments and tools have also been developed to significantly narrow the scope of editable features in blocks-based game development environments. This solves the problem of initially overwhelming the user with too many options by directing focus to a few specific parts. Code.org’s Code-Your-
Own FlappyBird tutorial serves as a good example of a bare bones game development environment.

Another way environments use restricted views is by embedding them within the full version. StarLogo TNG has a feature called “Subsets” that allows users to organize specific blocks, like most used blocks or blocks for a particular classroom activity, in custom drawers [8].

2.2 Gameblox

2.2.1 Overview

Gameblox is a new, visual, blocks based game development environment currently being developed by the Scheller Teacher Education Program (STEP) Lab at the Massachusetts Institute of Technology (MIT). Like many other blocks-based programming languages like Scratch and AppInventor, Gameblox is a browser-based tool, accessible through the web. Because Gameblox was designed for people with very little to no background in programming, users can create complex games by simply dragging objects onto specified screens, and attaching blocks of executable instructions. Users can create, modify, and test games all in one place.

Gameblox has two modes: an editor mode and a player mode. This thesis is primarily concerned with how a user interacts with the editor mode. The editor mode consists of three different views: design, blocks, and play. These views are also referred to as panels.

Gameblox is currently a web application built with Javascript. Django, a Python framework, is used on the server-side, and gives functionality to serve pages, handle uploads, and save games. Currently, MySQL is used for the database in the development phase, but Postgres is used in production.
2.2.2 Design

Overview

In the design panel, users can create, edit, and delete game objects, as well as change the look of their game. Users do so by dragging out and dropping game objects, called “entities,” into the central workspace called the “stage.” As of now, Gameblox supports the use of four different types of entities: sprites (characters), labels, text inputs, and sounds. Each entity type has classes, which share characteristics, functions, and restrictions, but can be edited to look and behave differently from each other. Once an entity is dragged out onto the stage, it becomes an “instance” of that particular entity. The view is divided into three parts: the lefthand panel, the stage, and a layer select panel.

Entity Panel

The lefthand panel consists of three main sections. First, the users can switch between entity types in the entity select panel. Second, in the classes panel below the entity select tabs, users can add, edit, delete, and switch between classes within the currently selected entity type. Third, in the side bar panel, users can edit various properties of their selected entity class or instance, or add their own custom properties. All classes that are created can be referred to in the blocks panel, regardless of whether or not there is an instance of the game object dragged into the workspace.

Stage

The stage is the central workspace to where all game objects are dragged. A game component is only included if it exists in the stage. Users can click on components in the stage to select them, which enables the editing of various component specific properties. Users can also move the game components around the stage through direct manipulation. The appearance and contents of the stage reflect where and what components will be displayed when the game is played.
Layer Select

The layer select panel, the rightmost column, also consists of three sections: selected component, background, and stage properties. First, the selected component panel displays information about whatever component in the workspace is clicked on. This information is similar to information displayed under the “Instances” tab in the sidebar panel. Second, in the background panel, users can update the background to a color or an uploaded image. Lastly, in stage properties, users can add new, load existing, delete, edit, and switch between different stages.

![Figure 2-2: Gameblox Design Panel](image)

2.2.3 Blocks

Overview

The blocks panel is where users interact with pieces called blocks to implement all of their game logic. Blocks consist of a shape, color, text, and some also have embedded dropdown menus. All blocks have affordances such as
notches and sockets to indicate where they can be connected to other blocks. Connecting blocks together allows the user to create larger programs.

Blocks are stored inside and organized under drawers. Each drawer contains a different number of blocks, and has a unique name and color associated with it. Drawers are organized under headers, which are used for organizational purposes only.

The Blocks Panel is split into three main sections: a tree of game components that reference work done in the Design Panel, a list of drawers and headers, and a blocks workspace.

**Design Panel Reference**

First, the leftmost column is a hierarchical display of components, which allows users to refer back to the game components they have created previously in the design panel without having to physically go back to the design panel. This column shows all created classes, and also instances dragged out into the design panel workspace. Users can edit names of instances and classes of components directly in the blocks panel if needed, which should also update the design panel view.

**Drawers and Headers List**

Next to the leftmost column is a list of headers and drawers. The list of drawers loosely groups all blocks by functionality, and headers loosely organize the drawers by purpose. Headers are denoted by a slot with a black background and white text, while drawers are denoted by a slot with a gray background, a small color square to the left, and black text. All blocks are initially hidden, and clicking on any given drawer displays all blocks allocated to that specific drawer. Drawers are highlighted in blue when they are selected. Only one drawer’s contents can be displayed a time.
Central Workspace

Lastly, users drag out, edit, and connect blocks from the drawers to the white workspace to integrate them into their game. At the bottom of the workspace, users can enable warnings and errors if desired. Users can also delete blocks by dragging them to the trash can icon in the workspace.

Figure 2-3: Gameblox Blocks Panel

Figure 2-4: Gameblox Blocks Panel (Events Drawer Open)
2.2.4 Play

The play view allows users to automatically and privately test their creations without having to save their work. The play view will execute the code within the current blocks workspace, and include the game components dragged onto the stage. Users can stop or reset their currently loaded game at any time.

Figure 2-5: Gameblox Play Panel
Chapter 3

Flexidor

3.1 Overview

3.1.1 Introduction

The flexidor is an alternative, restricted view of the Gameblox editor that only affects the design and blocks view of the editor mode. The name “flexidor” comes from the concatenation of the words “flexible” and “editor,” referring to the flexible nature of a flexidor design. In some figures and mockups, the flexidor may be referred to by its previous name, the “flexditor.”

3.1.2 User Interface

From a user interface standpoint, the flexidor was designed such that as many aesthetic elements of the user interface from the full Gameblox editor were retained as possible. In addition, all interactions remained consistent between the full editor and the flexidor. In this way, the design of the flexidor mimics the full editor in every way except for the number of features displayed.
3.1.3 Data Model

From a data model standpoint, the flexidor was designed such that it is integrated into the Gameblox game model itself as a property, with the key flexidors. This design decision enables cross-editor compatibility, meaning games created in one type of editor can be loaded, edited, and saved on another type of editor. In order to add further flexibility and leave room for future work, the flexidors property in the game model was designed as an array, which allows multiple flexidors to be saved. Each flexidor is represented as a flexidor object within the array. Currently, however, there is only functionality to support one flexidor per Gameblox game.

```javascript
game =
{
  assetImages: Array[Object]
  assetSounds: Array[Object]
  ...
  flexidors: Array[Object]
  ...
}
```

Each flexidor object has 5 properties: id, name, type, designPanel, and scriptsPanel. The id, name, and type are all automatically generated for every property in the Gameblox game model. In this case, the type is always “flexidor.” The designPanel and scriptsPanel properties are each objects that describe the restrictions necessary for each of their respective panels, and will be discussed further in the next sections. A complete picture of the full flexidor data model is included for reference.
{  
  id: int,  
  name: String,  
  type: "flexidor",  

  designPanel: {  
    included: boolean,  
    entityType: {  
      phaserPhysicsPiece: {  
        included: boolean,  
        newEntity: {  
          included: boolean,  
          sidebar: {  
            properties: { dictionary of boolean mappings },  
            instances: {},  
            traits: {}  
          }  
        }  
      }  
    }  
  }  

  entityIds: [ {{entityId_0}}, ..., {{entityId_n}} ],  
  existingEntities: {  
    {{entityId_0}}: {  
      included: boolean,  
      sidebar: {  
        properties: { dictionary of boolean mappings },  
        instances: {},  
        traits: {}  
      }  
    },  
    ...  
    {{entityId_n}}: {  
      included: boolean,  
      sidebar: {  
        properties: { dictionary of boolean mappings },  
        instances: {},  
        traits: {}  
      }  
    }  
  }  

  label: {  
    ...  
  },  
  TextInput: {  
    ...  
  },  
  rightPanel: {  
    included: boolean,  
    panels: []  
  },  
  scriptsPanel: {  
    included: boolean,  
    scriptsDrawers: [],  
    scriptsBlocks: {}  
  }  
}
A high level look at the flexidor data structure is also included below.

```javascript
flexidors: [{flexidor_0}, ..., {flexidor_n}]

flexidor_i =
{
    id : int,
    name: String,
    type: "flexidor",
    designPanel: {
        ...
    },
    scriptsPanel: {
        ...
    }
};
```

### 3.2 Design Panel

#### 3.2.1 User Interface

The granularity of the display is completely based on the granularity of the user’s restrictions. The primary idea is selective hiding and showing of the original, full Gameblox editor. If no attribute of the display panel is selected in the Flexidor Builder, the Design icon in the top navigation bar is hidden completely, and the user will only have access to the blocks panel. Entity types, classes, sidebar options, and features in the righthand column receive the same treatment. The icons representing these features are completely hidden if there is no corresponding selection, and users no longer have the option to select those hidden options.

Note that in the classes section, users have the ability to control whether or not new classes can be created, and also select a subset of features allowed for these newly created entities. Already created and newly created classes cannot be deleted in flexidors. Currently, the lowest level of specificity for entities is the ability to control which sidebar tabs (Properties, Instances, and Custom
Properties) appear. Individual properties can be selected in the Flexidor Builder, but will not individually appear in the corresponding flexidor.

The rightmost column can be completely collapsed if no relevant features are selected. The lowest level of granularity is the ability to show and hide the background, selected component, and state properties panels. The flexidor does not currently support individual form fields within the tabs in the rightmost column.

Figure 3-1: Example Flexidor Design Panel
3.2.2 Data Model Overview

Restrictions relevant to the design panel can be found in the flexidor data model, in the designPanel property saved as an object. Information is stored in multiple layers, getting more specific with each layer. At the top level, the designPanel object has three properties, included, which is a boolean designating whether or not the design panel is shown in the flexidor, entityType, an object that contains more details about restrictions pertaining to entities, and rightPanel, an object that contains more details about restrictions pertaining to the rightmost panel in the editor.

3.2.3 entityType

The entityType object contains three objects that represent the three main entity types: Sprites, referred to as the phaserPhysicsPiece object,
Labels, referred to as the label object, and Text Inputs, referred to as the textInput object. Each of these entity types has similar data models, and these data models correspond to restrictions for their respective entity type.

For each of these three types, there is a boolean called included that indicates whether or not the entity icon tab and panel is included in the flexidor, an object newEntity that describes restrictions for new classes within that entity type, an array called entityIds that lists existing entity id numbers in the saved game corresponding to entity classes, and also an object existingEntities that describes restrictions for any existing entity classes in the game. The entity IDs enumerated in the entityIds array correspond to the properties of the existingEntities object.

The newEntity object and objects within the existingEntities object are identical in structure. These objects have an included property that describes whether or not that specific class is included in the flexidor, and also a property called sidebar which enumerates further restrictions within the sidebar tabs. The sidebar object contains three properties that correspond to the sidebar tabs in the full Gameblox editor, properties, instances, and custom properties. The properties object is a mapping of entity class properties to booleans, set equal to true if included in the flexidor and false if not. Similarly, the instances object is a mapping of instance properties to booleans, and the traits object is a mapping of available custom properties to booleans.

An example of what the entity type object structure looks like is shown below.
phasPhysicsPiece: {
    included: boolean,
    newEntity: {
        included: boolean,
        sidebar: {
            properties: { dictionary of boolean mappings },
            instances: {},
            traits: {}
        }
    }
    entityIds: [ {{entityId_0}}, ..., {{entityId_n}} ],
    existingEntities: {
        {{entityId_0}}: {
            included: boolean,
            sidebar: {
                properties: { dictionary of boolean mappings },
                instances: {},
                traits: {}
            }
        },
        ...
        {{entityId_n}}: {
            included: boolean,
            sidebar: {
                properties: { dictionary of boolean mappings },
                instances: {},
                traits: {}
            }
        }
    }
},

3.2.4 rightPanel

The rightPanel object consists of two properties. First, the included boolean denotes whether or not to show the rightmost column in the design panel. Second, the panels array enumerates the names of the parts that are to be displayed. Choices included the selected component layer, the background layer, and the stage properties layer.
3.2.5 Integration

The implementation of the design panel of the Gameblox editor is not inherently conducive to easy customization of the view, as there is currently no underlying, existing data model defining which features exist on screen. Integrating the flexidor data model into the user interface required creating a few objects to temporarily store user interface element to data mappings, as well creating multiple disjoint functions to aid in rendering certain user interface elements. At times, as a last resort when no existing function had the desired behavior, custom show and hide functions were implemented.

3.3 Blocks Panel

3.3.1 User Interface

The default headers, drawers, and blocks are replaced by the user’s custom headers, drawers, and blocks. Headers are rendered exactly as they are rendered in the full Gameblox editor. The functionality and appearance of the drawers is identical, but the color of the blocks do not necessarily match the color swatch associated with a particular flexidor drawer. The click interaction with the drawers to open them, and the drag and drop interaction with the blocks remains constant as well.
Figure 3-3: Example Flexidor Blocks Panel (drawers closed)

Figure 3-4: Example Flexidor Blocks Panel (open drawer)
3.3.2 Data Model

The data model for the scripts panel only contains three properties. First, the included boolean describes whether or not to include the Blocks icon in the top navigation bar. Second, the scriptsDrawers array enumerates objects corresponding to drawer properties. Each drawer object is identical to a drawer object in the original Gameblox model. Third, the scriptsBlocks object describes what types of blocks are in each of the drawers listed in the scriptsDrawers array. This drawer to block object is identical in structure to the drawer to block object in the original Gameblox model.

```json
{
  included: false,
  scriptsDrawers: [],
  scriptsBlocks: {}
}
```
3.3.3 Integration

Integration of restrictions in the blocks panel was seamless. Like discussed in the previous section, the data model used to define a single drawer or header was identical to that used to define a single drawer or header in the full Gameblox editor. Similarly, the data model used to define drawer to blocks mappings in the Gameblox editor could also be used in the flexidor. Therefore, to correctly render the flexidor blocks panel, the only functions needed were those that converted UI elements into the correct format for drawers and blocks. Temporary data structures to keep track of UI information were created to facilitate referencing.
Chapter 4

Flexidor Builder

4.1 Design

4.1.1 Target User

The target user of the flexidor builder is someone who is already familiar with Gameblox, and wishes to create a simplified editor with a subset of the original functionality. Currently, both the edX course staff from the STEP Lab and the Gameblox team would fit this profile.

4.1.2 Preliminary Design

During the brainstorming phase, the two primary contenders for the flexidor builder interface were a form-based format and a visual overlay-based format.

The form-based format enumerated all choices textually, both in the design panel and blocks panel, with form fields and checkboxes in each form item. These form items were organized hierarchically and nested, such that users could expand and collapse them to reveal or hide options. The default form would be completely unselected.

The visual overlay-based format was an interface in which the user directly interacted with the existing Gameblox editor. An overlay containing
checkboxes were placed next to each selectable component directly on the editor, rather than a separate form. Users maintained the ability to flip between the design panel, blocks panel, and play panel.

Figure 4-1: Flexidor Builder Design Panel Form Mockup, with original name “Flexditor”

Figure 4-2: Flexidor Builder Design Panel Visual Overlay Form Mockup, with original name “Flexditor”
4.1.3 Final Design

Given that our target users are familiar with the Gameblox interface, the Flexidor Builder’s interface was ultimately designed to resemble the Gameblox editor as much as possible, and therefore both mimics the drag and drop behavior and also the editor’s appearance. The final design combines elements of both forms, by employing a separate form that simulates interacting with the full editor. Similar to the Gameblox editor, the Flexidor Builder is divided into a Design Panel form and a Blocks Panel form, and split further into the same subsections, with granularity down to entity properties.

4.2 User Research

An informal paper prototype design test of the Flexidor Builder was conducted early in the development stage in order to enforce and ensure usability and learnability principles before committing to an implementation. Because the target user group for the Flexidor Builder is people who are already familiar with the Gameblox Editor, I recruited users from the Gameblox development team. The initial design implemented and used in this test was the textual form-based format.

Two members of the Gameblox development team participated in this first iteration of user testing. Both members were prompted to complete the same specified task, and to voice their thoughts throughout the entire process. At the end of the test, participants engaged in an informal discussion about the user interface. No formal analysis was performed.

Following testing, it was concluded that a stand-alone form-based format was not an intuitive way to select features, and despite familiarity with different features of Gameblox, recalling what those features corresponded to on the editor was difficulty. Users often wanted to simulate the “drag and drop” motion enforced in the full Gameblox editor.
4.3 Overview

To create a flexidor, users must first create the full game they would like to restrict, and save it. From the full Gameblox editor, users can then launch the Flexidor Builder from the icon in the top navigation bar. The options displayed in the Flexidor Builder will reflect properties of the saved game.

Once the Flexidor Builder is launched, users are faced with four options: customizing the design panel, customizing the blocks panel, loading an existing flexidor, or saving progress via the “Done” button. Users cannot create an empty Gameblox editor; users must choose to restrict the design panel or blocks panel to create a new flexidor. Users also have the option of loading a flexidor they have already created into the form builder. By default, the form starts out as empty, and “Load Existing” populates the form with pre-existing values.

Figure 4-3: Flexidor Builder Home Screen
To restrict Design Panel features, users must select “Customize Design Panel” in order to view the Design Panel form. In order to select a subset of blocks, users must select “Customize Blocks Panel.” After filling out either of these forms, users must click the “Back” button to save their progress and return to the main screen. In order to hide a panel completely, users can choose to only customize one panel.

After users have finished selecting their restrictions, they must click “Done” and exit out of the overlay form. To complete the process, users must save the Gameblox game using the full Gameblox editor.

### 4.4 Design Panel Form

The design panel form in the flexidor builder very closely resembles the columns in the full Gameblox editor, minus the exclusion of the stage in the center. The form is visual in nature, and reuses the same icons, names, colors, interactions, and layout.

Users switch between entity types, classes, and sidebar tabs in the same way they would switch between entity types, classes, and sidebar tabs in the full flexidor. To select which components will be included in the resulting flexidor, users simply place a check in the checkbox next to desired components. Only selected components will be rendered in the resulting saved flexidor.

Tabs, such as the entity tabs and sidebar tabs, are not included in the subset of restrictable features, as they are affordances to the features that do exist. For example, an entity type tab is only displayed in a flexidor if a class of that entity type is selected in the Design Panel form.
4.5 Blocks Panel Form

4.5.1 Overview

In the blocks panel form, users have the freedom to organize and choose blocks in a manner that makes the most sense to them, and the game they wish to enable. Users can create custom headers and custom drawers, and then populate their drawers with blocks from a pre-populated blocks library organized into default categories. The blocks panel form relies on familiarity with the drag and drop nature and organization of the Gameblox editor interface. The blocks panel form is divided into four different parts, which are discussed in more detail in the following sections.
Figure 4-5: Flexidor Builder Blocks Panel Form

Figure 4-6: Flexidor Builder Example Blocks Panel Form Use Case
4.5.2 Drawer and Header List

The left-most column lists all current drawers and headers. Like the full Gameblox editor, headers are represented by slots with black background color and white text, whereas drawers are represented by a color, a gray background color, and black text. The current selected drawer is highlighted with a yellow border. The order of these headers and drawers is rearrangeable by drag and drop motion, and will reflect the order in which the headers and drawers appear in the completed flexidor. Headers and drawers can be deleted by clicking on the small red “x” at the right most edge of each individual slot. When drawers are deleted, its contents are also deleted. Open drawers cannot be deleted.

4.5.3 Drawer and Header Creation

The second section is at the very bottom, and is a mini-form where users create custom headers and drawers. Users must first select the slot type, and then enter an appropriate name. If the type is a drawer, users must also select a representative color. Slots cannot be created without filling in all the fields. Once created, the new drawer or header appears in the left column below the existing drawers and headers, and is ready to be populated with blocks. Note that users must first submit the drawer creation form before placing blocks inside.

4.5.4 Blocks Library

The rightmost column displays the full Gameblox block library, containing all blocks available to that particular game. The blocks library is organized by the default drawers, where each default drawer’s blocks are displayed on a different “page” of the library. The drawer name of the page is located underneath the box in capital letters. To flip through pages, users must click the left and right arrows next to the drawer name. Blocks from the Blocks Library can be dragged out of the
library and dropped into the current open drawer by grabbing the yellow rectangle at the end of each frame containing a block.

4.5.5 Drawer Blocks

Lastly, adjacent to the drawer column, the middle column outlined in white displays the contents of the current open drawer. A user must drag and drop blocks from the block library into the column in order to add it to the open drawer. Users may rearrange blocks within the drawer, also through a drag and drop motion. The order of the blocks within the selected drawer reflects the order in which the blocks will appear in the drawer in the flexidor. Lastly, users may remove blocks from the current drawer by dragging the blocks back into the block library.
Chapter 5

Evaluation

5.1 Flexidor

5.1.1 Study Design

I performed a comparative, controlled experiment in order to assess the usability of the two types of interfaces individually, and also to compare their usability. The study was designed to test the following hypotheses:

1. A flexidor is faster to use than a Gameblox editor.
2. A flexidor less frustrating to use than the Gameblox editor.
3. Therefore, a flexidor is easier to use than a Gameblox editor.
4. However, the use of a flexidor does not decrease learning or user satisfaction in game creation.

Throughout the study, researchers also recorded observations that were relevant to answering the following questions:

1. What are the common pain points when using either the Gameblox editor or a flexidor?
2. Where is there overlap and where is it disjoint?
In order to accomplish this, two treatments were administered and counterbalanced. Each treatment consisted of two tasks. Treatment Group A used a pre-designed flexidor to complete Task 1 and the full Gameblox editor to complete Task 2. Treatment Group B used the full Gameblox editor to complete Task 1 and a pre-designed flexidor to complete Task 2.

The independent variable for this study was the type of interface a user used, and the order in which the two interfaces were used. The dependent variables measured were speed, errors, and the user’s self-reported satisfaction. Speed and user responses to surveys were measured quantitatively whereas a user’s level of distress and error outcomes were measured qualitatively through observation.

Many confounding variables were taken into consideration, including familiarity with different operating systems, familiarity with different browsers, level of computing ability, and familiarity with blocks based programming languages. To control for some of these variables, the same laptop, a MacBook Pro ’13 Retina Display, using the same browser, Chrome, was used for all user studies. In addition, the tasks and order of the tasks administered was the same for all test subjects. Lastly, users were randomly sorted into the two treatment groups before testing.

<table>
<thead>
<tr>
<th>Order</th>
<th>Treatment A</th>
<th>Treatment B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>Flexidor</td>
<td>Full Gameblox Editor</td>
</tr>
<tr>
<td>Task 2</td>
<td>Full Gameblox Editor</td>
<td>Flexidor</td>
</tr>
</tbody>
</table>

Table 5-1: Treatment for Flexidor Study

5.1.2 Study Procedure

A total of 18 users, both men and women, were recruited through email lists and personal contacts, and were MIT undergraduate and graduate students
between the ages of 19 and 23. The 18 users were randomly split into 2 groups of 9, and were given the same laptop, pre-questionnaire, instructions, tasks, and post-questionnaires. Each trial for each task was screen and voice recorded. User trials consisted of 3 phases: a prequestionnaire, user briefing and tasks.

First, users took a pre-questionnaire evaluating their prior programming experience. Then, users were then given a briefing to read that quickly explained the motivation behind Gameblox, and given a short introduction on its main features. This user briefing also included study logistics, such as the purpose of the study and the voluntary nature of the study. Information included in the user briefing was reiterated out loud to reinforce content.

Second, users were told to complete their first task with their first assigned interface, in alignment with their randomly pre-assigned treatment group. They were first shown the specifications of the game they were meant to build, and walked through how a user would play the aforementioned game. Subjects were allowed to ask whatever questions they needed to before moving on to the instructions portion of the task. The instructions portion of the task outlined the steps needed to build the game, divided into a “Design” and “Blocks” category, and then further subdivided into steps. Again, subjects were allowed to ask questions before interacting with their treatment interface. Upon completion of the first task, users took the first part of a post-questionnaire which evaluated their reaction to the first interface, measuring their qualitative satisfaction.

Lastly, users were told to complete their second task with the other assigned interface, again in alignment with their randomly pre-assigned treatment group. The process of walking through specifications of the assigned was identical to that of the first. Upon completion of the second task, users took the second part of the post-questionnaire, which first evaluated their reaction to the second interface, as well as their reaction to the differences between the two interfaces.

Each task involved recreating a piece of a Gameblox game, and was carefully designed to explore different aspects of the Gameblox interface. Each
experiment took approximately 20 to 30 minutes to complete from beginning to end.

5.1.3 Data Analysis Method

Qualitative and quantitative data was collected through observations and through the post-questionnaires. These pieces of data are sorted into qualitative and quantitative in the table below, and further discussed in this section.

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Reported Programming Experience</td>
<td>Scale of 1 (less than 6 months), 2 (less than 1 year), 3 (1-2 years), 4 (3+ years)</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Self-reported Programming Level</td>
<td>1 (beginner), 2 (Intermediate), 3 (Expert)</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Time</td>
<td>Time it took to complete entire task in seconds</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Observations</td>
<td>General user reactions while using the interface</td>
<td>Qualitative</td>
</tr>
<tr>
<td>Self-Reported Sense of Accomplishment</td>
<td>Rating on a scale of 1 (none), 2 (low), 3 (medium) to 4 (high) on sense of accomplishment after completing a task</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Self-Reported Sense of Understanding</td>
<td>Score measuring the extent to which users understood the task they just completed.</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Self-Reported Difficulty (1st Task)</td>
<td>Scale of 1 (very easy) to 5 (very hard) of difficulty, both perceived and actual</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Self-Reported Difficulty (2nd Task)</td>
<td>Scale of 1 (very easy) to 5 (very hard) of difficulty, both perceived and actual</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Self-Reported Learning Between Tasks</td>
<td>Score assigned to the extent to which interface 1 aided in learning how to use interface 2</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Open-ended Comments</td>
<td>Free response section where user recalls moments that stood out to them.</td>
<td>Qualitative</td>
</tr>
<tr>
<td>Distress Indicators</td>
<td>Long pauses, aimless clicking, verbal frustration</td>
<td>Qualitative</td>
</tr>
</tbody>
</table>

Table 5-2: Data Collected for Flexidor Study

A flexidor is faster to use than a Gameblox editor.

There are a few different time differences that may indicate that a flexidor is faster to use than a Gameblox editor. First, within treatment groups, the average time difference between the two successive tasks could possibly suggest a difference in speed. However, a learning effect could also be the cause of the difference.
Second, between treatment groups, the average time difference of the same task between different interfaces could also possibly suggest a difference in speed. However, randomization may not have accounted for differences in programming ability between users revealed in the pre-questionnaire. Both results will be discussed.

A flexidor is less frustrating to use than the Gameblox editor.

The self-reported sense of understanding, and low numbers and severity of distress indicators may indicate that an interface is less frustrating to use. Insights taken from observations such as outcomes of, types of, and reactions to errors may also prove useful.

A flexidor is easier to use than a Gameblox editor.

The previous comparisons, in conjunction with self reported scores for difficulty may suggest that one interface is easier to use. The types of errors observed and the locations of error occurrences during task completion may also give insight to pain points in each type of editor. Errors that do not occur while using one interface over the other may suggest an advantage in that area, and vice versa.

There are a couple of differences in difficulty scores that may be of interest. First, one could measure the differences between perceived and actual difficulty of one task within one treatment group. One could also compare these metrics between the two tasks. Second, one could measure the differences between perceived and actual difficulty between treatment groups but on the same task. Here, the perceived difficulty would hypothetically be identical because users are given the same task. Differences in actual difficulty may indicate that one editor is easier to use. Third, one could measure the differences between perceived and actual difficulty of one interface between treatment groups.
A flexidor does not decrease learning or user satisfaction in game creation.

Self reported learning, sense of accomplishment, and sense of understanding are all factors that could contribute to learning or user satisfaction. Analyzing differences in task completion times between tasks may also be useful.

5.1.4 Data Results

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Programming Level</td>
<td>2</td>
<td>1.6</td>
</tr>
<tr>
<td>Average Programming Experience</td>
<td>3.33</td>
<td>2.33</td>
</tr>
</tbody>
</table>

Table 5-3: Answers to Pre-Questionnaire
(see scale in Table 5-2)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Perceived Difficulty</td>
<td>1.2</td>
<td>1.4</td>
</tr>
<tr>
<td>Average Actual Difficulty</td>
<td>2.8</td>
<td>3</td>
</tr>
<tr>
<td>Task 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Perceived Difficulty</td>
<td>1.6</td>
<td>2.1</td>
</tr>
<tr>
<td>Average Actual Difficulty</td>
<td>3</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Table 5-4: Average Self Reported Difficulty
(1 Very Easy – 5 Very Hard)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Time (sec)</td>
<td>369</td>
<td>588</td>
</tr>
<tr>
<td>Average Understanding (1 none – 5 high)</td>
<td>4.6</td>
<td>4.4</td>
</tr>
<tr>
<td>Average Accomplishment (1 low – 4 high)</td>
<td>3.1</td>
<td>3.2</td>
</tr>
<tr>
<td>Task 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average Time (sec)</td>
<td>394</td>
<td>90</td>
</tr>
<tr>
<td>Average Understanding (1 none – 5 high)</td>
<td>4.6</td>
<td>4.6</td>
</tr>
<tr>
<td>Average Accomplishment (1 low – 4 high)</td>
<td>3.4</td>
<td>2.4</td>
</tr>
</tbody>
</table>

Table 5-5: Post-Questionnaire Data
The first interface was easier to use.

The first interface was more enjoyable to use.

The first interface was more fun to use.

The first interface helped me learn how to use the second interface.

<table>
<thead>
<tr>
<th>Question</th>
<th>Flexidor First</th>
<th>Full Editor First</th>
</tr>
</thead>
<tbody>
<tr>
<td>The first interface was easier to use.</td>
<td>2.9</td>
<td>1.3</td>
</tr>
<tr>
<td>The first interface was more enjoyable to use.</td>
<td>2.4</td>
<td>3.3</td>
</tr>
<tr>
<td>The first interface was more fun to use.</td>
<td>2.9</td>
<td>3.6</td>
</tr>
<tr>
<td>The first interface helped me learn how to use the second interface.</td>
<td>4.6</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Table 5-6: Average Self Reported Comparisons
(1 Strongly Disagree – 5 Strongly Agree)

5.1.5 Common Errors

<table>
<thead>
<tr>
<th>Error Description</th>
<th>Flexidor Frequency</th>
<th>Full Editor Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes text of both labels instead of just one</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Does not connect blocks because believes they will execute top to bottom</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Does not know where to find blocks (does not know to click on drawers)</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Wants to connect blocks horizontally</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Believes label text and variable to be synchronized if same name</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Drags out text input in place of label</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Looking for variable setting in design panel</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Clicking empty space to type in 0 instead of dragging block into slot</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Dragging out math blocks to increment variable by 1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Dragging out incorrect blocks to set variable to 0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Drags out sensing block to detect click</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Selects variable block to change label</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Defines procedure call</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Uses “pointing in direction” block under direction and looks for “left block.”</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Uses a degree instead.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drags out unused segmented blocks for functions, such as “is left arrow pressed”</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>“sprite’s x speed to 0”</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5-7: Error Descriptions and Frequencies
5.1.6 Discussion

Looking at post-questionnaire data in the previous section, time differences both within treatment groups (between tasks) and across treatment groups (same task) seem to support our hypothesis that the use of the flexidor is faster than the use of the full Gameblox editor. In all of these comparisons, subjects spent a shorter amount of time using the flexidor. The flexidor designed for Task 1 was the fastest combination, resulting in an average completion time of 90 seconds. Within the same treatment group, this is 6.5 times faster than completing Task 1 with the full Gameblox editor, and when compared with the other treatment group, this about 4 times faster than completing Task 2 with the full Gameblox editor.

Looking at the quantitative data in Table 5-7, the frequency and severity of errors incurred while using the full Gameblox editor is greater than that while using the flexidor. In general, from qualitative observations, subjects exhibited fewer outward distress signals while using the flexidor, especially in Task 2. Task 2 confused none of the subjects in Treatment B were confused by Task 2. In addition, levels of understanding reported in Table 5-5 are similar between treatment groups and tasks. This data and these observations may suggest that flexidors are less frustrating to use than the full Gameblox editor.

Average self reported understanding scores (Table 5-5) seemed similar between all tasks and treatment groups, and average self reported learning comparison scores (Table 5-6) between treatment groups indicate that both groups thought whichever interface they used first helped them learn how to use the second. This seems to suggest that the use of a flexidor either does not affect learning or is similar to the learning outcome of using the full editor. This also seems to suggest that the functionality and interactions learned while using the two different types of interfaces are transferrable. We did not see a decrease in average completion time within Treatment Group A between tasks.

Looking at the average self-reported actual and perceived difficulty (Table 5-4), in combination with average responses to the comparison questions in the
post-questionnaire (Table 5-6), we cannot conclusively say that the flexidor is easier to use, or that the flexidor provides the same sense of accomplishment derived from completing a game in the full Gameblox editor. While results may be inconclusive for ease of use, and accomplishment, some interesting points of consideration arose from user testing that are worth discussing:

**Flexidor Design**

Before user testing, one flexidor was created for Task A and a separate flexidor was created for Task B. These flexidors were created in a way that I thought made the most sense logically. However, both flexidors were different in design. The flexidor for Task A had blocks grouped by instruction order. The flexidor for Task B had blocks grouped by function.

Several issues arose because of this difference in thinking. Some subjects thought that the flexidor in Task One was labeled in a way that made the headers and drawers seem like instructions rather than part of the editor interface. To new users who were not familiar with the interface, this was confusing because they could not find where the blocks were. In the full game editor, the headers have the word “Blocks” contained in them, which avoids this misunderstanding. In addition, the way the blocks were grouped in the flexidor was not necessarily always the way subjects were thinking about the block logic. In some cases, users were thrown off by block groupings, and preferred looking through the default drawers because of it.

These observations emphasize the importance of flexidor design, and the dangers of a poorly designed flexidor. Having fewer features presented to the user could be helpful but is not necessarily better, as different interfaces for different audiences to fit their needs was found to be very important. Because the subset of features was so specific, the flexidor forces users to think in a certain way that they might not be used to. This, in turn, may make it more difficult and less enjoyable for them to learn or use Gameblox, and may have affected the self-reported scores in the post-questionnaires. Therefore, users’ previous knowledge
or experience, their sensibilities, purpose, and goal are all factors to be considered while designing flexidors.

Background of Participants

All subjects were either current or recent graduates of MIT, a school that prides itself on challenging and pushing students to stretch themselves and to work their hardest. Many members of the MIT community view challenges as enjoyable, which may not be representative of the greater Gameblox community. This type of culture may affect subjects’ satisfaction and accomplishment ratings. In addition, all students had some amount of programming background, and nearly half of subjects had 3+ years of programming experience. Performing these or similar tasks with a group of subjects with none to little programming experience may yield different results. At times, subjects’ prior knowledge of programming aided them greatly in classifying blocks and programming logic, as well as understanding the relationships between entities, classes, and instances.

Error Outcomes

Because subjects were not given a tutorial of the Gameblox editor prior to the user test, users used some portion of Task 1, which sometimes continued into Task 2, to learn how to use their given editor. Some subjects took this a step further and explored different sets of functionality not specified in the directions. One of three outcomes occurred when a subject committed an error.

First, some errors were lapses in knowledge of certain editor mechanics, but the interface gave immediate visual cues to make it obvious to the user that he or she had made a mistake. In this case, users knew they were doing something incorrectly right away, and were actively looking for the correct solution. For example, some users tried clicking on an empty space in a block, but saw that nothing happened. After realizing that the system was not responding in the way they wanted, many realized the hole needed to be filled by
a block and could not be directly set through keyboard input. These lapses were bridged after subjects used a trial-and-error approach to solving their original problem. Because subjects usually learned quickly from these types of errors, they were more exploratory in nature.

Second, some errors were a result of misunderstanding a piece of the editor, of which there was a delayed indication that the user needed to try again. Users would not know they had committed an error immediately, but the system would very obviously indicate the error at a later time, which then directed users in looking for the correct solution. For example, users needed to play and test their game to recognize that the variable score and the text “score” were not connected. After testing the game, users would see that the label’s text was set to the word “score” and not the variable, prompting them to then change the block to the variable block rather than the text block.

Third, some errors were conceptual mistakes in programming logic, which tunnel-visioned users into looking for something very specific. In these cases, users became “stuck” as there was no clear indication that they had committed a conceptual error, so they usually continued to explore the same path. For example, many subjects were fixated on finding a “left” block for Task 2 and became fixated on the “Direction” drawer. They could not think outside the box and connect changing direction to flipping an image rather than the literal changing of direction.

Whereas the first and second type of error was seen throughout both types of interfaces, the third type was mostly seen in Task 2 with the full Gameblox editor. This framework of classifying Gameblox usability errors in conjunction with the error analysis done in Table 5-7 may suggest that the use of a flexidor does not prevent all errors, but may be useful in addressing a specific subset of possible mistakes.
5.2 Flexidor Builder

5.2.1 Study Design

I performed a usability study of the Flexidor Builder tool. The study was designed to test the following hypotheses:

1. The Flexidor Builder is easy to use.
2. The Flexidor Builder is easy to learn.

Observations relevant to answering the following questions were recorded throughout the study:

1. What are the most common mistakes subjects use when using the Flexidor Builder?
2. What functionality do users wish the Flexidor Builder had?

5.2.2 Study Procedure

A total of 6 users, both men and women, were recruited through email lists and personal contacts, and were MIT undergraduate and graduate students between the ages of 20 to 28. The 6 users were given the same laptop, pre-questionnaire, user briefing, task, and post-questionnaire. User trials consisted of 5 phases: a pre-questionnaire, a user briefing, a tutorial, building an example game, and lastly, a task. Each trial of the assigned task was screen and voice recorded.

The pre-questionnaire administered in this study was identical to the pre-questionnaire administered in the flexidor user study. The user briefing included the same logistics, but in addition to the motivation behind Gameblox, the briefing also contained a short motivation and explanation of a flexidor, and introduced the concept of a Flexidor Builder.
After the briefing was read through, researchers would step through a short tutorial of the Gameblox editor. The Design Panel and Blocks Panel were discussed in succession, and each UI element in both panels were pointed out and explained. Interactions, such as drag and drop, additions, and deletions were also explicitly mentioned. Lastly, Gameblox naming conventions were described to subjects.

After the tutorial, users were asked to build a pre-specified Gameblox game using the full Gameblox editor. They were given the same instructions and specifications as Task 2 in the previous study. Users were allowed to ask whatever questions they wanted and needed to complete the task. Researchers were allowed to intervene if the subject became stuck during a step.

Lastly, users were asked to recreate a flexidor using the Flexidor Builder. Subjects were given three color screenshots of the flexidor displaying the flexidor design panel and the flexidor blocks panel, and asked to voice aloud their actions and thoughts as they were navigating the feature. The Flexidor Builder was pre-launched for them at the start of the task.

5.2.3 Data Analysis Method

Again, qualitative and quantitative data was collected through observations and through the post-questionnaires. These pieces of data are sorted into qualitative and quantitative in Table 5-8, and further discussed in this section as they relate to the initial hypotheses and questions.
A high self-reported ease of use score for both the flexidor design panel form and the flexidor blocks panel form, combined with a low number and severity of distress indicators may suggest that the Flexidor Builder is easy to use.

A high sense of self-reported sense of understanding, a high self-reported learnability score, and a high level of confidence of being able to complete a similar task using the Flexidor Builder may indicate that the Flexidor Builder is easy to learn.

### 5.2.4 Data Results

All scores obtained during the post-questionnaire phase were rated on a scale of 1 to 5, where 1 is strongly disagree and 5 is strongly agree with the provided statements. All users completed the task.

<table>
<thead>
<tr>
<th>Data</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Observations</td>
<td>Qualitative</td>
</tr>
<tr>
<td>Self-Reported Ease of Use</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Self-Reported Sense of Understanding</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Self-Reported Learnability</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Matching Behavior Expectation and Reality</td>
<td>Quantitative</td>
</tr>
<tr>
<td>Open-ended Comments</td>
<td>Qualitative</td>
</tr>
<tr>
<td>Distress Indicators</td>
<td>Qualitative</td>
</tr>
</tbody>
</table>

Table 5-8: Data Types for Flexidor Builder Study

<table>
<thead>
<tr>
<th>I found this system easy to use.</th>
<th>It was easy for me to get the system to do what I wanted it to do.</th>
<th>I understood how to use the Blocks Panel Form.</th>
<th>I understood how to use the Design Panel Form.</th>
<th>I knew how to navigate around the feature.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.17</td>
<td>4.33</td>
<td>4</td>
<td>4</td>
<td>3.83</td>
</tr>
</tbody>
</table>

Table 5-9: Ease of Use Responses
I would feel comfortable using this feature again to accomplish another similar task. It was easy for me to learn how to use the Blocks Panel form. It was easy for me to learn how to use the Design Panel form.

<table>
<thead>
<tr>
<th>I learned how to use the feature quickly.</th>
<th>It was easy for me to learn how to use the Blocks Panel form.</th>
<th>It was easy for me to learn how to use the Design Panel form.</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>4.83</td>
<td>4.5</td>
</tr>
<tr>
<td>4.5</td>
<td>4.33</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-10: Learnability Responses

<table>
<thead>
<tr>
<th>Number of Subjects</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completed Task with 0 Errors</td>
<td>4</td>
</tr>
<tr>
<td>Completed Task with 1 Blocks Panel Error</td>
<td>1</td>
</tr>
<tr>
<td>Completed Task with 1 Design Panel Error</td>
<td>1</td>
</tr>
<tr>
<td>Completed Task with More Than 1 Error</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5-11: Completion Statistics

5.2.5 Question Results

What are the most common mistakes subjects use when using the Flexidor Builder?

The opt-in nature of the Flexidor Builder design panel form was not immediately clear to some subjects, and they were confused about how checking a checkbox would correspond to the final flexidor. Adding a line of instruction within the editor may alleviate this problem. Creating a real-time preview feature that would allow users to see how Flexidor Builder changes affect the flexidor UI may also alleviate this problem.

In the Blocks Panel form, a common point of confusion was the order of drawer creation versus the placing of blocks within drawers. Approximately 50% of users attempted to drag blocks into the drawer first, and then submit the drawer form. This resulted in further confusion as the blocks disappear as they are dropped in the supposed “drawer.” The need to press “Create” on the form to create a drawer before populating the drawer with blocks was not clear, as some users thought “Create” was final. Most subjects read from top to bottom, and because the form is at the
bottom, this layout may have suggested a different order than intended. Once subjects continued exploring and created a drawer, however, they quickly were able to recreate the action.

**What functionality do users wish the Flexidor Builder had?**

In the Design Panel form, almost every user expressed a desire for a preview feature, to let them know how they are progressing, and to provide feedback for their actions. Many users asked if there was a side-by-side preview that would respond to the checkboxes they were clicking available to them.

In the Blocks Panel form, users most frequently attempted to drag the full body of the block, as they do in the Gameblox editor Blocks Panel, but did not seem frustrated that they could only grab onto the end of the piece. Most users thought the identical drag and drop motion was still intuitive. However, many subjects wanted to edit drawer properties, like the name, after creation by clicking on the drawer directly.

Overall, subjects also wanted a more explicit signal that their work was being saved. Currently, they had to assume that their work was being saved while navigating in between forms, and wanted a more explicit “Save Progress” or “Save” in place of a “Back” button.

**5.2.6 Discussion**

The quantitative data gathered seems to suggest that the Flexidor Builder was both easy to learn and easy to use. However, it appears that most users thought the feature was easier to learn than to use. Indeed, observations gathered during the studies confirmed this, as many asked questions about their current progress and were hesitant of their responses throughout the task, but were reassured about their assumptions upon completion. Many of the observations recorded matched and re-emphasized comments made in the open response
section as well. However, because the sample size was so small, more user testing would need to be done to make our results conclusive and statistically significant.
Chapter 6

Future Work

6.1 Properties

Currently, the most specific part of the Gameblox editor a user can restrict via the Flexidor Builder is content within the sidebar panels in the Design Panel. Users can select individual properties. However, the most specific area that a flexidor can display is one level higher, as a flexidor will render the entire sidebar panel rather than specific properties selected. Implementing the final step requires extra infrastructure such as getters and setters for the UI elements, which do not currently exist.

6.2 Loading Existing Flexidors

Being able to continue editing a flexidor design is a lot more efficient than having to start over every time a user launches the flexidor builder. Loading an existing flexidor back into the Flexidor Builder requires implementing a few data model to UI to temporary data model mappings.
6.3 Multiple Flexidors

Supporting multiple flexidors would allow a user to create a succession of editors with differing amounts of features, which could potentially be used and distributed as a ramp-up to the full editor, or for students of differing programming backgrounds. This feature in combination with the ability to load existing flexidors enables users to easily and efficiently create a “set” of flexidors that build upon each other.

6.4 Preview

Visual feedback is an important usability principle. Implementing a preview tool within the Flexidor Builder would allow users to automatically, and easily see if their actions are corresponding to intended purpose. A preview function would make recognizing and correcting an error much faster and easier, and also increase learnability. An alternative or addition could be implementing a more clear affordance for selection into the flexidor, like fading the image when not selected or adding a feedback message when a user checks a checkbox.

6.5 Procedure Calls

Custom procedures that can be used as black-box functions may lower the cognitive load and barrier for programming education, by increasing the readability of blocks of code. Currently, the blocks library only contains default blocks. However, in the future, including procedure calls defined by the user into the blocks library would be an incredibly powerful feature.
6.6 Flexidor Builder Usability Improvements

User testing revealed several possible areas for usability improvements. Rearranging the layout of the Blocks Panel Form, adding clearer labels and instructions throughout, and providing more explicit affordances would all improve the current user experience of the Flexidor Builder and make the flexidor creation process more intuitive. Allowing for multiple selection, including a select all method, and adding a search tool would increase efficiency for experienced flexidor users.
Chapter 7

Contributions

This thesis first motivated, and then defined a flexidor as a Gameblox editor with a restricted feature set. It then detailed the design and implementation of the user interface, backend, and data model of a flexidor within the Gameblox editor. In the process, a data model to describe the design panel of a Gameblox game also emerged. In addition, this thesis defined and detailed the design and implementation of the user interface and backend of the Flexidor Builder, a tool within the Gameblox editor used to create flexidors.

This thesis also presented detailed designs, procedures, data, and analyses of studies conducted on both the flexidor and the Flexidor Builder. Though the use of flexidors was found to be both faster and less frustrating, results and observations from these studies revealed the central impact of flexidor design on ease and enjoyment of use in particular. Careful consideration of the organization of blocks when creating flexidors was found to be especially significant. The programming background, familiarity with the Gameblox editor, and the end goal of the flexidor user are all factors that play an important role.

The ideas in this thesis will become a foundation on which future work around adding flexibility and customizability to blocks based languages can be built upon. The designs, implementations, and observations in this thesis can be further expanded upon in the context of online game creation, outside of Gameblox.
Bibliography


Appendix A

Pre-Questionnaire

Email *
Please provide primary email address. This will only be used if I need to follow up with you on something regarding the study.

Age *
(in years)

What kind of laptop do you own? *
- Mac
- Linux
- Windows
- Other

What is your preferred internet browser? *
What do you use most often?
- Chrome
- Safari
- Firefox
- Internet Explorer
- Other

How familiar are you with programming? *
(In terms of time)
- Not familiar
- <6 months
- <1 year
- 1-2 years
- 3+ years

How would you rate your programming proficiency? *
- None
- Beginner
- Intermediate
- Expert

How familiar are you with blocks based programming languages? *
These include, but are not limited to, Scratch, AppInventor, Tale Blazer, StarLogo
- Not familiar
- <6 months
- <1 year
- 1-2 years
- 3+ years

How would you rate your proficiency in blocks based programming languages? *
- None
- Beginner
- Intermediate
- Expert

How would you rate your proficiency in Gameblox? *
- I have never used Gameblox
- Beginner
- Intermediate
- Expert
Appendix B

User Briefings

Flexidor: Briefing for Test Users

Introduction
Gameblox is a blocks based programming environment akin to Scratch, where users are able to drag and drop game objects to add characters to their game, and drag and drop blocks instead of writing code to add game logic.

The purpose of this study is to compare two different designs of the Gameblox editor; you will be given both of these designs. My goal is to evaluate the usability of each given interface at each point in time throughout the task. Your feedback in the tasks will provide me with valuable information, which will potentially impact the future direction of Gameblox.

Logistics
Throughout the exercise please remember we're testing the system; we're not testing you. If you have any trouble with the assigned task, keep in mind, it is likely the system's fault. Any raw information we collect will be strictly confidential, meaning none of your individual results will be shared publicly. This study is completely voluntary, and if at any point during the study you feel uncomfortable, you are free to stop the test and leave.

Task 1
Gameblox users use the editor to create games. Your first task is to recreate a piece of a Gameblox game, both visually and behaviorally, with your given editor. Please feel free to ask as many questions as you need to in order to complete this task.

Task 2
Your second task is to recreate another piece of the Gameblox game, both visually and behaviorally, with the second editor. Again, feel free to ask as many questions as you need to in order to complete this task.

Flexidor Builder: Briefing for Test Users

Introduction
Gameblox is a blocks based programming environment akin to Scratch, where users are able to drag and drop game objects to add characters to their game, and drag and drop blocks instead of writing code to add game logic.

The purpose of this study is to assess the usability of a specific feature in Gameblox called the Flexidor Builder. My goal is to evaluate the usability of this feature interface throughout the task. Your feedback in the tasks will provide me with valuable information, which will potentially impact the future direction of Gameblox.

A Flexidor is a Gameblox editor with only a subset of the features available via the full Gameblox editor. Users can restrict visibility of different components when designing their Flexidor via the Flexidor Builder.

Logistics
Throughout the exercise please remember we're testing the system; we're not testing you. If you have any trouble with the assigned task at any time, keep in mind, it is likely the system's fault. Any raw information we collect will be strictly confidential, meaning none of your individual results will be shared publicly. This study is completely voluntary, and if at any point during the study you feel uncomfortable, you are free to stop the test and leave.

Tutorial
Let's explore some of the functionality Gameblox has! I will now walk you through the Gameblox interface. Please stop me at any time if you have questions.

Example Game
You will now build 1 example game. Please follow the specifications and instructions given to you.

Task
Now, you will get the chance to create your own flexidor! Recreate the pictured flexidor as accurately as possible. Please walk through your actions, and tell me what you think you should be doing at all points in the task. When you think you are done, just say aloud, "I'm done."
Appendix C

Tasks

Task 1: Click To Score

User sees gem and instructions, as well as initial score set to 0.

Whenever user clicks the gem, the score increments by 1. User sees cumulative score at all times.

Instructions

Design
1. Drag out 2 labels onto the editor.
2. Change the text of one label to "Click to Score!" and the text of the other label to "0"

Blocks
1. Initialize a variable called "score" to 0.
2. Drag out blocks such that when Ruby is clicked, score is incremented by 1, and the label "0" is set to the updated score.
Task 2: Movement and Background Color

User sees stationary character upon initialization.
User sees a non-white background color.

Whenever user presses left button, character moves to the left, and faces to the left.
Whenever user presses right button, character moves to the right, and faces to the right.

Instructions

Design
1. Drag out a sprite onto the screen
2. Change the background color to a color of your choice.

Blocks
1. Drag out blocks such that when left arrow is pressed, the sprite is flipped to the left and its x speed is set to -100
2. Drag out blocks such that when right arrow is pressed, the sprite is flipped to the right and its x speed is set to 100,
Flexidor Specs: Design Panel

Blocks Panel Part A

Blocks Panel Part B
Appendix D

Post Questionnaires

Flexidor

Post-Questionnaire Part I
Thank you for participating in the study! Please complete after finishing the task.

* Required

Please re-enter your email address *

What was your first task called? *
☐ User Test
☐ User Test B

How difficult did you perceive this task to be? *
1: Very Easy 2: Easy 3: Medium 4: Difficult 5: Very Difficult

Very Easy ☐ ☐ ☐ ☐ ☐ Very Difficult

How difficult was it to actually complete the task? *
1: Very Easy 2: Easy 3: Medium 4: Difficult 5: Very Difficult

Very Easy ☐ ☐ ☐ ☐ ☐ Very Difficult
Rate your sense of accomplishment upon task completion. *
1: None, 2: Low, 3: Medium, 4: High

1 2 3 4

None ☐ ☐ ☐ ☐ High

Please rate your understanding of the program you created. *
1: Do not understand, cannot reproduce
2: Vaguely understand, would have trouble reproducing
3: Understand well enough to reproduce
4: Understand, can definitely reproduce

1 2 3 4 5

Do not understand, cannot reproduce ☐ ☐ ☐ ☐ ☐ Master, can reproduce and would be able to expand

Were you frustrated at any point in time? If so, explain. *

Please write any questions or comments you have about this specific task and interface.

Post-Questionnaire Part II
Thank you for participating in the study! Please complete after finishing the task.
* Required

Please re-enter your email address *

What was your second task called? *
☐ User Test C
☐ User Test D

How difficult did you perceive this task to be? *
1: Very Easy 2: Easy 3: Medium 4: Difficult 5: Very Difficult

1 2 3 4 5

Very Easy ☐ ☐ ☐ ☐ ☐ Very Difficult

How difficult was it to actually complete the task? *
1: Very Easy 2: Easy 3: Medium 4: Difficult 5: Very Difficult

1 2 3 4 5

Very Easy ☐ ☐ ☐ ☐ ☐ Very Difficult
Rate your sense of accomplishment upon task completion. *
1: None, 2: Low, 3: Medium, 4: High

Please rate your understanding of the program you created. *
2: Vaguely understand, would have trouble reproducing 3: Understand well enough to reproduce 4: Understand, can definitely reproduce

Were you frustrated at any point in time? If so, explain. *

Please write any questions or comments you have about this specific task and interface.

The first interface was easier to use. *

The first interface was more enjoyable to use. *

The first interface was more fun to use. *

The first interface helped me learn how to use the second interface. *

Please write any questions or comments you may have comparing the two editors.
Post-Questionnaire

Thank you for completing the study!

* Required

Please re-enter your email address *

I knew how to navigate around the feature. *
Was it easy to know where to go and how to get there?

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree

I learned how to use the feature quickly. *

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree

I would feel comfortable using this feature again to accomplish another similar task. *

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree

I found this feature easy to use. *

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree

It was easy for me to get the system to do what I wanted it to do. *

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree

Actual system response and behavior matched my expectations. *

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree

I understood how to use the Blocks Panel form. *

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree

I understood how to use the Design Panel form. *

1 2 3 4 5

Strongly Disagree ☐ ☐ ☐ ☐ Strongly Agree