Kodename

Project Functional Specifications

Joe Ancona
Stephen Chung
Miracle Okubor
Benjamin Salah
Isaac Tyan

Rutgers University - Department of Computer Science
CS431 Software Engineering (01:198:431:01)
Instructor: Alex Borgida

Functional Summary

In recent years, the world is beginning to realize the importance of computer programming and people are either going back to school to learn computer programming, or teaching themselves how to program through books, videos and tutorials on the internet. According to forbes.com the top job for 2013 was Software Developing. This is one of many instances of the importance of computer science, especially computer programming, in the world today. While it is impressive and inspiring to see members of the older generations in computer science 101 courses in universities worldwide, it would be more impressive to get the younger generation, specifically children, engaged and interested in computer programming.

There have been attempts to engage children in programming, like MIT's *Scratch* which is an event-driven application with a fun, interactive user interface and sprites that simulate the instructions in your program. Another example is Stanford University professor, Richard E. Pattis's *Karel* which is an educative programming language used to create instructions that a robot called Karel uses to navigate, and solve problems within, its World. The Karel language is similar in structure and syntax to existing programming languages and so gives the young programmer an idea of what programming is like in general. However, since Karel, like any other programming language, requires valid syntax to create programs, there are obvious challenges with teaching young children the techniques of programming while avoiding the nuisance the memorizing syntax. Scratch syntax, however, is more conversational and is organized into code pieces much like a jigsaw puzzle. Unlike Karel, the user needs no prior knowledge to put together "puzzles pieces" and then begin solving problems. Unfortunately Scratch itself is not without problems because its syntax is substantially different from common mainstream programming languages.

Kodename is an feature-rich application developed by the Kodename team that aims to combine and improve upon the best features of Karel and Scratch for the purposes of teaching children in a fun and interactive manner the fundamentals of programming. This includes learning some pillars of programming such as modularity, reusability, and other skills that can be easily applied for any other programming languages they may encounter in the future. Kodename attempts bridge the application gaps present in Karel and Scratch by both avoiding the need to understand deep language syntax while still creating a programming vocabulary bears resemblance to other mainstream languages.

Kodename uses the Karel programming language to teach the user programming and with a user interface, Kodename aims to eliminate the need for the user to focus on syntax. The Kodename application allows the user to create a World for the Karel robot based on specifications determined by the user. The Karel robot's World is a grid/maze-like environment that contains the Karel robot, walls which the robot cannot walk through, and beepers which the robot can either pick up or set down. Once a World has been created, the user is presented with the interface that has Code Blocks that contain Karel language instructions, a Simple View panel to show the Code Blocks and a World which is the World the user created at the start of the application. The application allows the user to place Code Blocks into the Simple View; these Code Blocks are the chosen instructions that will help the user solve problems related to the World they have created. Once the user is done and they run their program they see the Karel robot move within the World and they can also see how each movement is related to some Code Block previously placed within the Simple View.

These are some basic features of the Kodename application. Others include:

- Creating custom functions that allow the user to divide their program and easily reuse these functions instead of repeating multiple lines of code.
- Editing and deleting custom functions.
- Removing Code Blocks from the Simple View.
- Clearing ALL the Code Blocks previously added to the Simple View.
- Undoing recent actions in the Simple View. Such actions include adding a Code Block, deleting a Code Block or clearing the Simple View.
- Continuous execution of the users program.
- Step-by-Step execution of the users program.
- Displaying status messages during execution within a console.

The underlying operating environment for the Kodename application is JAVA, using features from the Java Swing library to build the user interface. The application is not suitable for mobile devices such as smart phones, smart watches and tablets. It is suitable for desktops and laptops. After the initial software download, the Kodename application does not require a connection the internet to work.

Use Cases

Use Case: Setup Wizard

Purpose: To start the program and reach the Main Screen. Before the Main Screen is shown, the user will choose to load a previous session or begin a new one.

Normal case: World is loaded or created. If a world is loaded, any associated Karel code from the previous session using that world is also loaded. The user is placed at the Main Screen.

Prerequisites: The user must have a save file if she wishes to load a previous session.

Scenario A: The program opens to a welcome screen that explains to the user their choices (load a session, or start a new session). There is one button for each choice. Clicking on Load will bring the user to Scenario B. Clicking on New Session will bring the user to Scenario C.

Scenario B: A file chooser dialog will open. This will contain an address bar showing the user's current directory in the file system, a pane showing the contents of the current directory, a field for typing in the file name, and a drop down menu to filter by file type. It will contain an Open button that is active when a file is selected and a Cancel button that is active at all times. If Open is clicked, a session will be loaded and the user will be sent to the Main Screen (see next use case). If Cancel is clicked, we return to Scenario A.

Error Cases

1. A file that is not a valid save file is selected, or a corrupt file is selected. Resolution: Display error message and return to file chooser.

Optionally, the file chooser dialog may also contain buttons for the following actions: up one directory, home directory, toggle list / icon view.

Scenario C: A new window opens where the user can edit a world in a WYSIWYG fashion. The default world is 4 x 4 and contains no beepers. Karel sits at the bottom left corner, facing North. He can be dragged and dropped onto any location in the grid. There exists an arrow with the tail end at Karel and the head indicating the direction he is facing. This arrow can also be dragged and dropped to change Karel's orientation.

Above the main WYSIWIG editor is an action bar with the buttons Open World, Save World, and Clear. There are also two text fields for setting the size of the world. If the user changes these dimensions, then the world size will change accordingly. Karel will be reset to the bottom left position facing North and all other items in the world will be removed.

The Open World button launches a file chooser dialog (described earlier) that loads an existing world and allows it to be edited.

Error Cases

1. A file that is not a valid save file is selected, or a corrupt file is selected. Resolution: Display error message and return to file chooser.

The Save World button launches a file chooser that allows the user to save her current work to disk. The Clear button removes all walls and beepers, and returns Karel to the origin.

To the left of the editor is a widget bar from which graphical icons representing wall segments and beepers reside. The user can drag and drop wall segments and beepers into the grid, so long as no more than one object resides in each square (and Karel does not coincide with a wall).

Error Cases

1. The user tries to drop an object into a location that is already occupied.

Resolution: Display error message briefly.
Note that Karel is allowed to share a location with exactly one beeper.

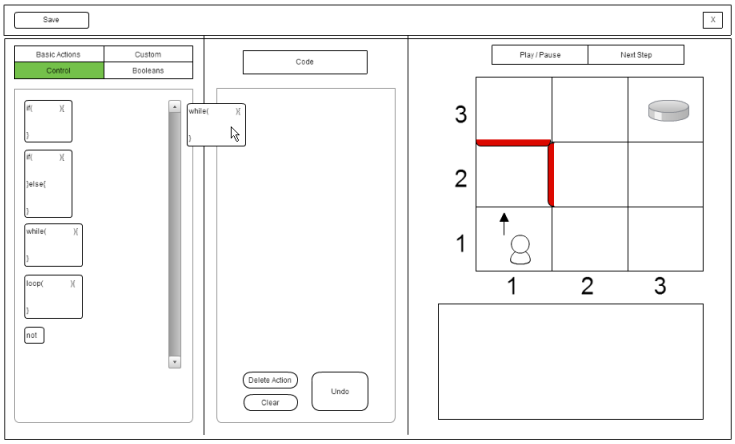
To the bottom of the editor are the buttons Ok and Cancel. Clicking Ok takes the user to the Main Screen with the world as shown. Clicking Ok returns the user to Scenario A.

Use Case: Adding, and creating new, Code Blocks

Purpose

- Add Code Blocks
- Create new Code Blocks (Custom Code Blocks)
- Edit Custom Code Blocks

Prerequisite: The user has generated a World environment by either creating themselves or randomly generating it.



The user is dragging a while loop and preparing to drop it into Code View

Scenario A: Adding Code Blocks in the Main Window

In order to create a program, the user will wish to add some Code Blocks. In our basic design proposal, the user will add a Code Block by double clicking on a Code Block from the leftmost pane of Kodename. This will then append this new Code Block to the end of the existing set of Code Blocks in Code View which is the middle display pane where the user's Code Blocks are displayed. For a normal/fancy version of Kodename, the user will be able to drag and drop Code Blocks anywhere in Simple View. Complexities arise when the user attempts to add a control-flow statement such as an if-statement. For a basic version of Kodename, when the user attempts to add an if-statement by double-clicking, a separate prompt will then appear which guides the user through populating the various components of an if-statement such as the conditional and the body. Once the user finishes walking through this prompt, the finished conditional appears in Code View. For our normal to fancy version of Kodename, the user drags and drops an empty if-statement into Code View. The user then drags and drops appropriate Code Blocks, e.g. conditionals and general functions, into the if-statement Code Block that is currently residing in Code View.

Scenario B: Create new Custom Code Blocks (Custom Code Blocks)

Although Kodename provides the user with adequate Code Blocks to solve the problem(s) presented in the World, the user may nevertheless wish to create Custom Code Blocks which allow the user to interact with the World with less, more simplified, code. To accomplish this, the user navigates to the "Custom" code tab above the left pane and clicks "Add a Custom Function". After the Custom Code Blocks window opens up, the user then selects various Code Blocks from the left pane and, using mechanics identical to how they add Code Blocks in the Main Window, adds it to the pane on the right. If the user adds a Code Block that they then wish to delete, they can select and highlight that particular Code Block and then click "Delete Action". This removes that Code Block from the pane. If the user then changes their mind and wishes to reverse their code deletion, the user can click "Undo" which will reverse the last user action. When the user is satisfied with their new custom Custom Code Block, they will then give this Custom Code Block a name and click "Save". After the user has clicked "Save", Kodename will check to see if the Custom Code Block the user has created is syntactically valid, e.g. there is not an if-statement where its conditional section is populated with a non-boolean Code Block. If there are syntax errors, Kodename generates an error message dialog box for the user forcing them to go back and revise their code. The user can also choose to forsake any work they've done for this Custom Code Block by clicking "Cancel". Once the user's

code is valid, the user clicks “Save” and Kodename will close the Custom Code Blocks window and bring the Main Window back into focus. The new Custom Code Block the user created will now show up under the “Custom” tab.

Scenario C: Edit/Delete Custom Code Blocks
The user selects a Custom Code Block from under the “Custom” tab by clicking it. The user then clicks “Delete” to delete this Custom Code Block entirely. A confirmation dialog box appears to make sure this is what the user wants to do. Similarly, the user can click “Edit”, while a Custom Code Block is selected. This brings up the Custom Code Block window. The user can then edit the Custom Code Block by either deleting or adding new code Code Blocks. When the user is satisfied with their editing, they can save their changes for this Custom Code Block by clicking “Save”. As with when they first created this Custom Code Block, Kodename will verify that there are no syntax errors. If there are, the user will be shown an error dialog and be forced to return and make appropriate changes. The user can also simply choose forsake all their changes by clicking “Cancel”. Once the user clicks “Save” for a syntactically correct Custom Code Block, the Custom Code Block window will close and bring the Main Window back into focus.

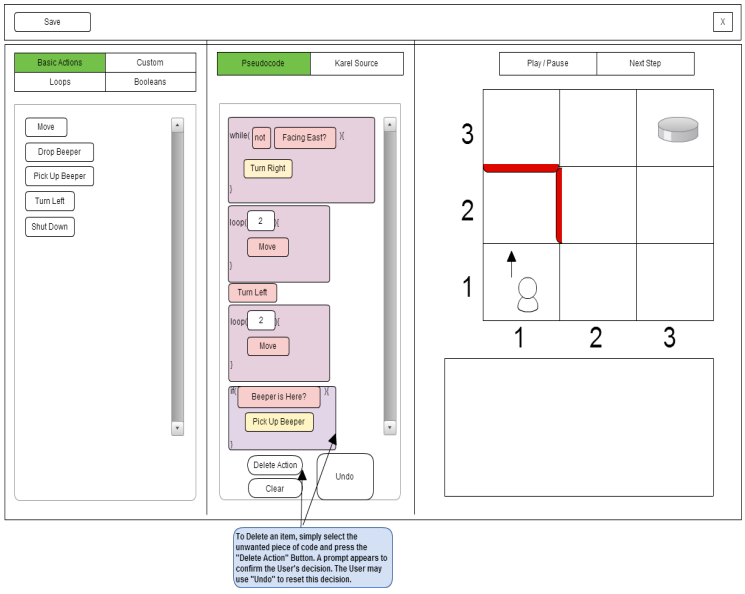
Error Cases:

1. If the user tries to add invalid code, this will generate an error. For instance, if the user attempts to add a “move” function in the exit condition of a loop, this will throw an error since the loop statement expects to see some kind of boolean statement.
2. If the user attempts to save a Custom Code Block that contains syntax errors, Kodename will display an error dialog.

Use Case: Deleting, Undoing, and Clearing Code

Purpose

- Delete Code
- Undo Action
- Clear Entire Code

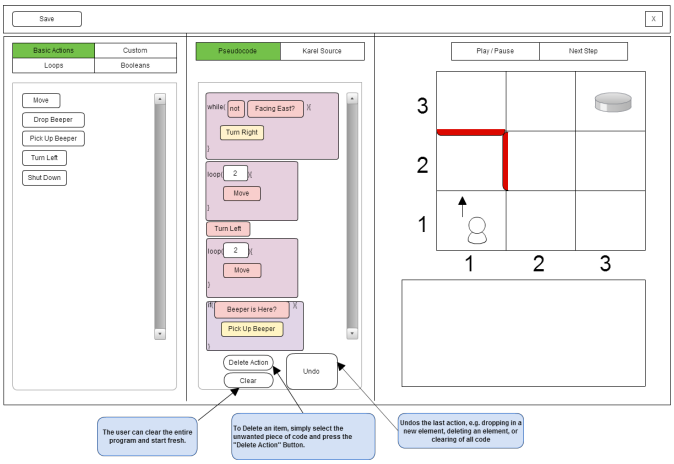


Prerequisite: The user has generated a World environment by either creating themselves or randomly generating it.

Scenario A: Delete Code
Kodename allows users to delete the code that they write in the event of a mistake. In all systems (Kernel, Normal, and Fancy), a delete is made by selecting a piece of code, clicking the “Delete Action” button and then clicking yes on the confirmation prompt. The difference is that in the Normal System, users can delete individual statements inside of conditionals as opposed to the Kernel System that only allows complete code block deletion. Once confirm is clicked, the selected code is removed from the code panel. If the user does not want to delete, they click the no button when the confirmation dialog appears. If the deletion of code was a mistake, then the user can click the “Undo” button which will revert the program to its state prior to the delete. If the user clicks the “Delete Action” button and there is no code in the panel, a message will appear in the console to notify the user that there is nothing that can be deleted.

Scenario B: Undo Action
In Kodename, users have the ability to quickly undo their last action. This feature is not available in the Kernel System. In the Normal System, users can undo their most recent action. By clicking the “Undo” button, they can reverse their last action with the change appearing in the code panel. Since this is only a single action undo operation, if the user presses the “Undo” button once more, the action that was originally undone will be applied to the program once again, therefore negating any

undo operation. In the Fancy System, users can make use of the multiple action undo which allows users to go all the way back to the original blank program. This might be useful for when an if statement is created and some statements are added to the inside of the code block. If the user decides that there is a better way to code this scenario, they can click the “Undo” button until the statements inside and the actual if statement disappear. When a clear operation is applied and the entire code panel becomes blank, users can undo this start-over attempt in the Fancy System. In the Normal or Fancy System if no code has been added or is in the code panel and the “Undo” button has been clicked, a message will appear in the console that reports to the user that there is nothing to undo.



Scenario C: Clear Entire Code

Kodename understands that sometimes it is easier to start fresh than to dissect and debug large amounts of code. We give users the operation to clear everything that is in the code panel and start fresh. This feature is not available in the Kernel System, but is fully supported in the Normal and Fancy System. Simply click the “Clear” button, then click yes on the confirmation dialog and the code will disappear. If no is clicked, the code will remain present. If clearing the code was a mistake, users can undo a clear action only in the Fancy System. If there is nothing in the code panel and the “Clear” button is clicked a message will appear in the console that tells the user that there is nothing to clear.

Use Case: Clearing the Simple View (the “Clear” button)

Purpose

- To remove all code blocks from the Simple View and start over

Prerequisite: User has added some Code Blocks to the Simple View

Scenario A: The user clicks the “Clear” button

The “Clear” button is clicked when the user wants to refresh their project. Once the button is clicked, a prompt appears to confirm the users’ decision. The prompt describes the consequence of the clear button, to be sure the user understands the action they are about to take. The prompt has two buttons, “No” and “Yes”. When the “No” button is clicked, the user is returned to the main window and NO Code Blocks have been removed from the Simple View. The “No” button cancels the action. When the “Yes” button is clicked, the user is returned to the main window and ALL Code Blocks have been removed from the Simple View. The “Yes” button approves the action. In a fancy version of our system, if the user approves the clearing action, the “Undo” button can be used to reverse the decision.

Use Case: Deleting a Code Block from the Simple View (the “Delete” button)

Purpose

- To delete a code block from the Simple View

Prerequisite: User has at least one Code Block in the Simple View.

Scenario A: The user clicks the “Delete” button

The “Delete” button is clicked when the user wishes to edit their instruction choices by deleting a Code Block that they had previously added to the Simple View. Once the button is clicked, a prompt appears to confirm the users’ decision. The prompt contains a simple message and two buttons, “No” and “Yes”. When the “No” button is clicked, the user is returned to the main window and the intended Code Block is not removed form the Simple View. The “No” button cancels the delete action. When the “Yes” button is clicked, the user is returned to the main window and the intended Code Block is removed.

Use Case: Running - Play, Pause, Next Step

Purpose:

- Play through your code automatically
- Pause the running of your code
- Play through your code step by step

Prerequisite: The user has generated a world environment by either creating themselves or randomly generating it. The user has produced code blocks into the code pane.

Scenario A: Pressing the Play button
The user can run his or her program by pressing the Play button, where the Play button will then act as a Pause button for later use. Kodename will begin to execute the program written and tracing will appear starting from the first block of code. The current block of code being traced will reflect the current change in our world unless it is a condition that cannot be shown. The next logical block of code in our code pane will be be traced after a three second delay timer where again the code being traced will reflect the current change in our world. In the fancy mode an input field will be provided next to the play button if the user wishes to increase or decrease that three second timer.

Kodename will have eventually executed, traced and reflected upon in the world all of the program's code blocks with either a success/unsuccessful message OR the program will stop at a specific code block with an error message due to an invalid action reflected in the world. In the normal mode a status message will be displayed in the console at every action that is being executed, traced, and reflected upon in the world. In the fancy mode the user will have the option of letting Kodename check and display invalid code block. Invalid states can occur for example if Karel can only move through two more cells without turning and the user exceeds that. Similarly if the user specifies actions that would lead Karel into a wall where the Karel tries to keep moving past the wall, the program will end.

The user has the option to press the pause button at any time while the program is running. Pressing the pause button will result in Scenario B.

Scenario B: Pressing the Pause button
The user can pause his or her program by pressing the Pause button, where the Pause button will revert back to begin a Play button for later use. Kodename will halt any further executing, tracing, and changes to our world until the user again presses the Play button or the Next Step button. The console will display a message letting the user know that the program is paused. The user can press the Play button in which Kodename will resume the program from where it was left off using its three second delay timer. The user can also press the Next Step button which will result in Scenario C.

Scenario C: Pressing the Next Step button
The user can manually step through his or her program once paused. Each time the button is pressed, the next logical block of code in our code pane will be be executed, traced, and reflected upon in the world. Kodename will have eventually executed, traced and reflected upon in the world all of the program's code blocks with either a success/unsuccessful message OR the program will stop at a specific code block with an error message due to an invalid action reflected in the world. In the normal mode a status message will be displayed in the console at every action that is being executed, traced, and reflected upon in the world. In the fancy mode the user will have the option of letting Kodename check and display invalid code block. The user has the option to press the Play button at any time which will result in a similar scenario to that of Scenario A.

Error Cases:

1. Attempting to run a program with no code block in the code pane
2. Pressing the delete action, undo, or clear button while running
3. Adding blocks while running
4. Pressing the next step button while in playing mode
5. Invalid moves reflected in GUI such as moving out of the grid, walking past a wall.

Use Case: Saving the current user session

In the Kernel version of the system, save functionality is not enabled. Every session is a single session and cannot save a current session or upload a previous session to continue work on. In the Basic system, the save functionality is enabled.

Purpose

- To be able to return to previous Kodename programs and continue solving a world.
- To keep a record of all Kodename programs for future reference

Prerequisite: The user has added at least one Code Block to the Simple View

Scenario A: The user has added some Code Blocks and is saving for the first time
Once the user has begun to add Code Blocks to the Simple View, in the process of solving their world, the user can choose to save current state of his program. The user does so by clicking the "Save" button. Once the button is clicked, a prompt is given to the user. The prompt contains a text field, where the user can enter a name for their program, and two buttons, "Cancel" and "Save". When the "Cancel" button is selected, the system returns the user to the main window without saving the program. When the "Save" button is selected, the system saves details of the users' Code Block choices and the details of the world created by the user(fancy version), randomly generated by the system(Basic/Normal version) or selected by the user(kernel version). The system saves these details in a file under the name entered by the user in the text field.

Error Case: An error will occur if the user does not enter a name in the text field but clicks the "Save" button. The system will give an error notification, explaining the reason for the error and reminding the user they need a name to save their project. The system returns the user to the prompt generated by the "Save" button clicked from the main window.

Scenario B: The user has added some more Code Blocks after the initial save and would like to save again.
After the initial save, the user continues with their program and after a while the user would like to save their updated state. The user does this by clicking the "Save" button again. Once the button is clicked, the save prompt is presented to the user, this time; the text field contains the name the program was initially saved as. The user can change the name if the wish. When the "Cancel" button is selected, the system returns the user to the main window without saving the current state of the program. When the "Save" button is selected, the system saves details of the users' Code Block choices and details of the world in the World section of the main window. In the fancy version of the system, after the initial save action, the system automatically saves every change made to the session.

Use Case: Exiting the Application

Purpose

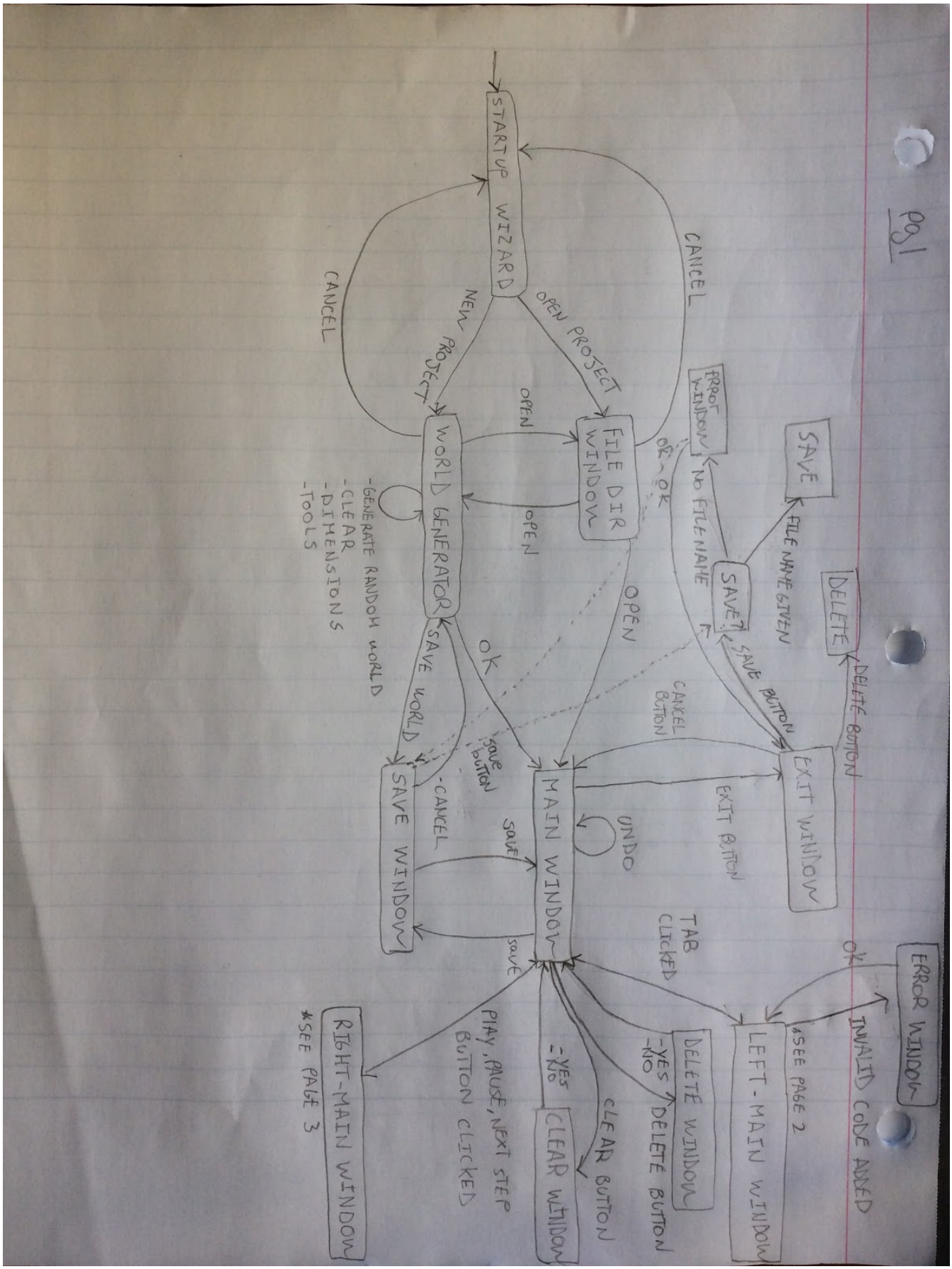
- To safely end the Kodename application and close the user interface.

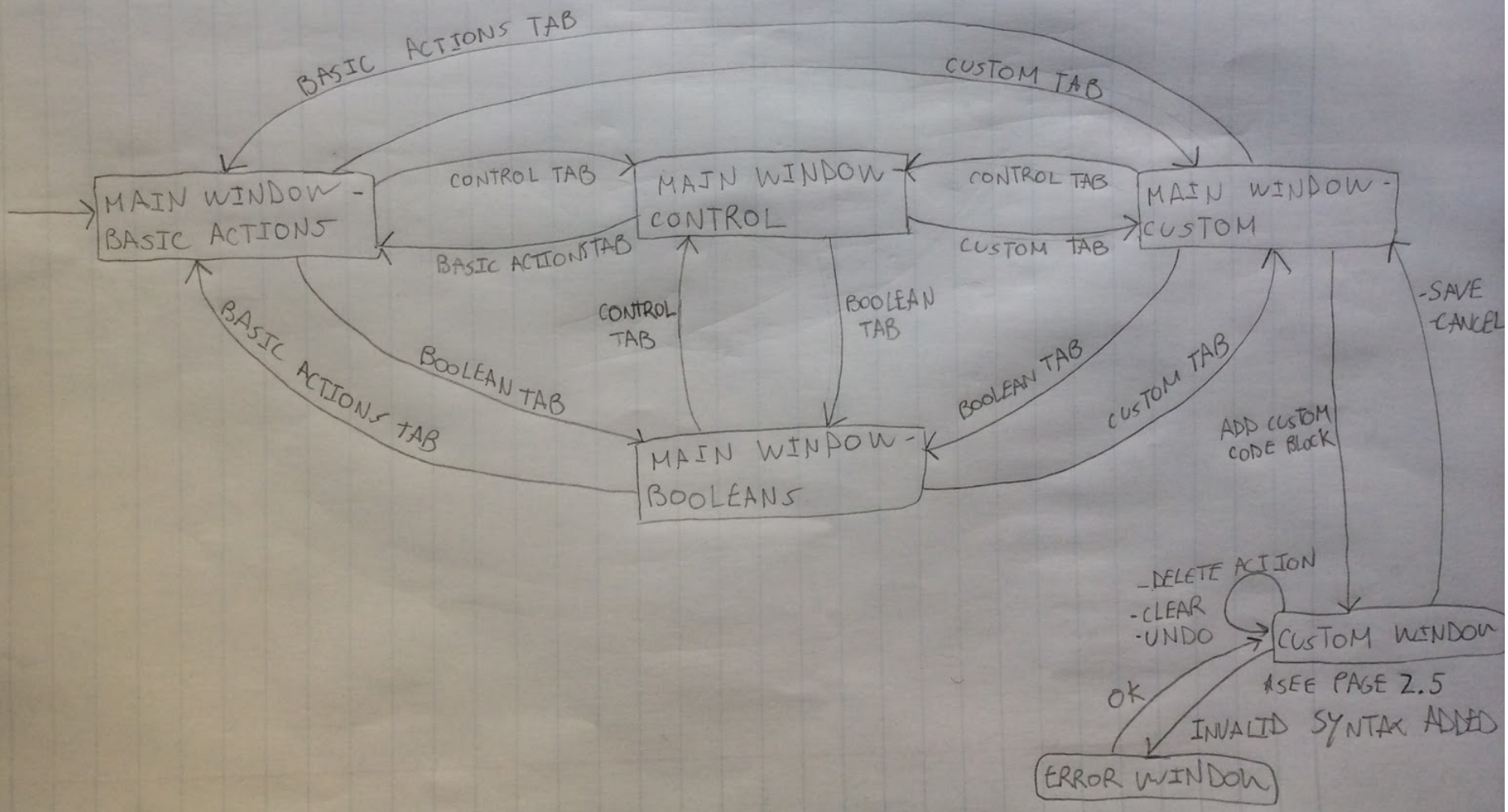
Prerequisite: There user is in the main window of the program.

Scenario A: User wishes to close the application
When the user is in the main window, the user can close the application at anytime whether they have added Code Blocks to the Simple View or not. The user closes the application by clicking the "X" button at the top right corner of the main window. When the user does this, the system makes sure there are no processes currently running in the window, that is, the system checks that the users program is not running, if it is, it stops the running process. In the Kernel system, the application gives the user a prompt to confirm that they want to exit the application. The prompt has two buttons, "Cancel" and "Exit". The "Cancel" button returns the user to the main window and the "Exit" button closes the application. There is no error case in the Kernel system for this scenario. In the Basic system, with the added functionality of saving the users' program, after the application has stopped the programs execution, if it was executing, the application gives the user a prompt to give the user the option to save the current state of their program or delete the program before exiting. The prompt has a text field for the user to enter a name for their program. If the user has saved a past state of the current session, the name used to save past states will be in the text field. If the user has not saved any past states the text field will be empty and the user will enter a name there. The prompt also has three buttons, "Delete", "Cancel" and "Save". The "Cancel" button returns the user to the main window. The "Delete" button closes the application without saving the current state, and, if previous states of the users' session had been saved, it deletes all files corresponding to the current session. The "Save" button saves the current state of the users program and details of the users world under a file with the name specified by the user before it closes the application.

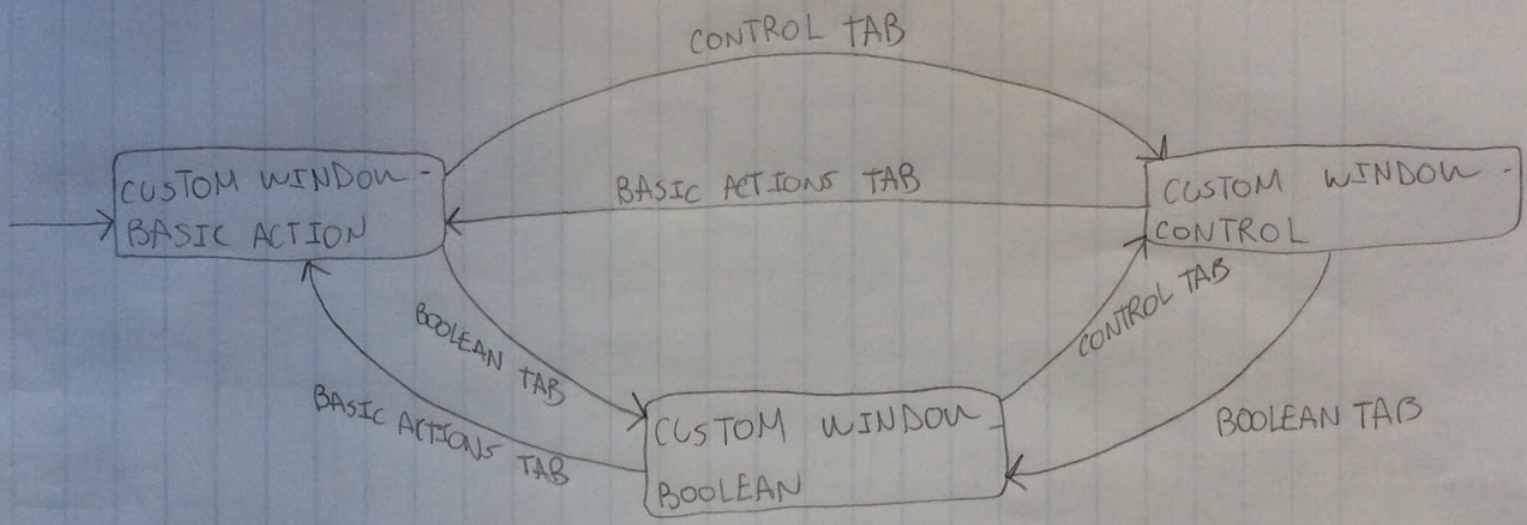
Error Case: An error occurs if the user does not enter a name in the text field but clicks the "Save" button. The system will give an error notification, explaining the reason for the error and reminding the user to enter a name for their project. The system returns the user to the prompt generated by the "X" button.

FSM Diagrams

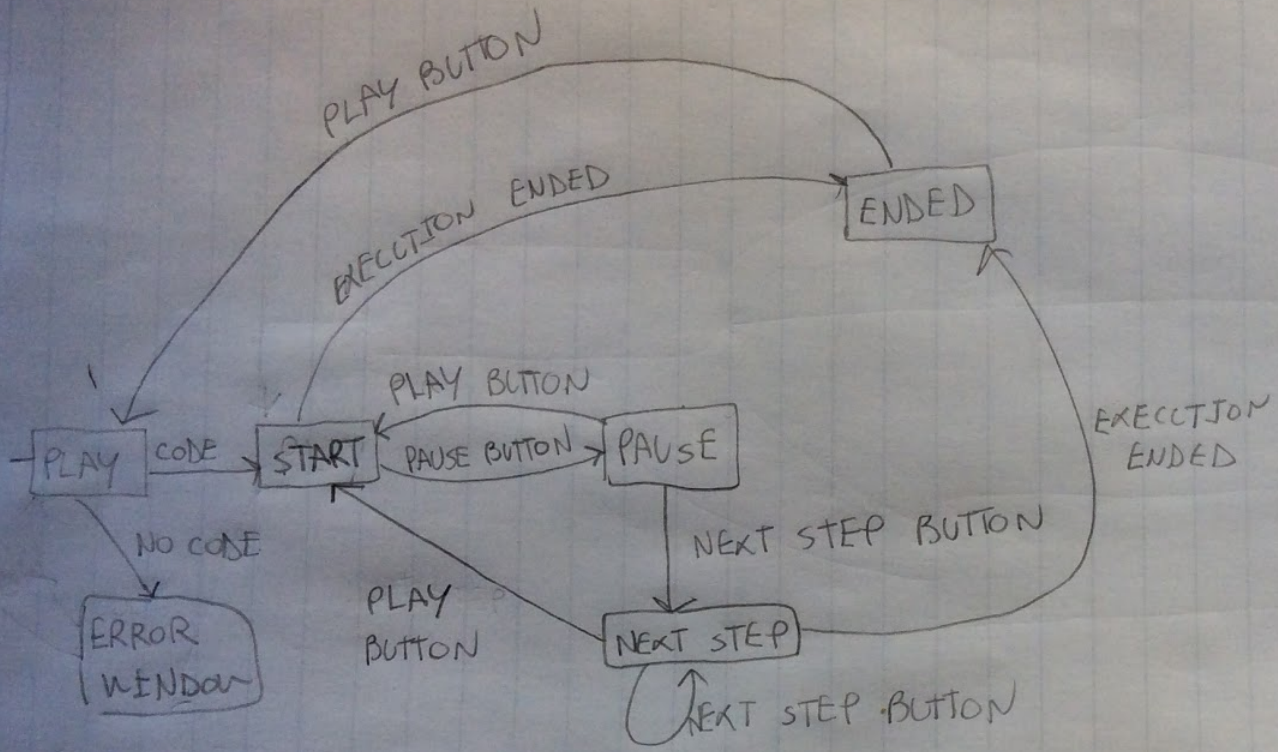



BASIC ACTIONS TAB
CUSTOM TAB
MAIN WINDOW - BASIC ACTIONS
CONTROL TAB
MAIN WINDOW CONTROL
CONTROL TAB
MAIN WINDOW - CUSTOM
BASIC ACTIONS TAB
CUSTOM TAB
CONTROL TAB
BOOLEAN TAB
BASIC ACTIONS TAB
BOOLEAN TAB
BOOLEAN TAB
CUSTOM TAB
MAIN WINDOW - BOOLEANS
ADD CUSTOM CODE BLOCK
-SAVE
-CANCEL
-DELETE ACTION
-CLEAR
-UNDO
CUSTOM WINDOW
*SEE PAGE 2.5
OK
INVALID SYNTAX ADDED
ERROR WINDOW

Pg 2.5


CONTROL TAB
CUSTOM WINDOW - BASIC ACTION
BASIC ACTIONS TAB
CUSTOM WINDOW - CONTROL
BOOLEAN TAB
CONTROL TAB
BASIC ACTIONS TAB
CUSTOM WINDOW - BOOLEAN
BOOLEAN TAB

PG3


PLAY BUTTON
EXECUTION ENDED
ENDED
PLAY BUTTON
PLAY
CODE
START
PAUSE BUTTON
PAUSE
EXECUTION ENDED
NO CODE
NEXT STEP BUTTON
ERROR WINDOW
PLAY BUTTON
NEXT STEP
NEXT STEP BUTTON

Karel Language Specification

Backus-Naur Form

<start> ::= TurnOn(); <morestmts> TurnOff();
<morestmts> ::= <stmt> <morestmts> | ε
<stmt> ::= <atomic> | <control>
<atomic> ::= Move(); | PickBeeper(); | PutBeeper(); | TurnLeft();
<control> ::= <if> | <iterate> | <while>
<if> ::= if <condition> { <morestmts> } | if <condition> { <morestmts> } else { <morestmts> }
<iterate> ::= iterate(<number>) { <morestmts> }
<while> ::= while <condition> { <morestmts> }
<condition> ::= not <predicate> | <predicate>
<predicate> ::= frontIsClear(); | leftIsClear(); | rightIsClear(); | facingNorth(); |facingSouth(); |
facingEast(); | facingWest(); | nextToABeeper(); | anyBeepersInBeeperBag();
<number> ::= <digit> | <digit> <number>
<digit> ::= 1 | 2 | 3 |4 |5 |6 |7 | 8 | 9 | 0

Feasibility

From a very high-level perspective, Kodename has two primary components. The first is the Setup Wizard which allows the user to generate a World by either creating one themselves or selecting to randomly generating one. The second main component of Kodename is what we have dubbed the “Main Window”. This is where most of the user interactions with Kodename will take place. Here, the user will actually create, edit, run, and trace a program that they create in order to solve a problem in the World they generated.

One of the first challenges we must tackle is the Setup Wizard. Since we plan in our fancy version to allow the user to create and edit their own World, we will have to design some type of sandbox map editor which will allow the user to populate the World with beepers, walls, and any other objects we deem as appropriate. Whether this map editor feature is feasible or not largely hinges on whether Java Swing supplies the appropriate API for such a task. Based on our current findings, we believe that the Swing Drag and Drop API may prove to be adequate but this will require further testing.

Our normal version of Kodename simply generates a random World for the user. This means that the user has no influence on what the World looks like or what it consists of. The only criteria for a randomly generated World is that it must have a “solution”. For instance, the robot cannot be surrounded by walls on all sides when the World is initialized. Finally, our basic version gives the user a set of pre-made Worlds to select from. Implementing both the normal and basic versions of Kodename, as far as World generation is concerned, is feasible provided we design an algorithm which can test whether a World has a solution or not. Our current strategy is to treat the World as a maze and ensure that every beeper is accessible and that the maze has an exit. The World will have to be its own Java class that contains beeper and wall objects as well as a set of functions which various state-related information such as the number of beepers it currently contains. At the very least, if our normal version for World generation proves to be too difficult, we will fall back to our basic version implementation which simply gives the user a set of pre-made Worlds to choose from. These Worlds in the basic version are guaranteed to have a solution and will be easy to create.

The second primary component of Kodename is the Main Window where the user actually creates programs to solve various problems presented by the World. Code Blocks are probably one of the most important objects for Kodename. One design strategy is to have an abstract class represent a generic Code Block and then have subclasses extend this abstract class to represent specific types of Code Blocks such as booleans, control flow statements, e.g. if/else statements and loops, and

general functions (including Custom Code Blocks). This architecture will hopefully allow us to have better management over our code by applying the principles of abstraction and data encapsulation.

Another key feature of Kodename is a function which checks whether a user-written program, or Custom Code Block, is syntactically valid and if it will "compile". This algorithm can be thought as a compiler of sorts and will run in the background without the user explicitly calling it into action. For instance, when the user saves a Custom Code Block, this compiler function must be called to check if this new Custom Code Block will actually run without hitting any syntax related errors. An example syntax error is if, while reading the exit condition of a loop, the compiler comes across a "move" function, instead of a boolean Code Block.

As with World generation, the extensibility of this compiler feature hinges on which version of Kodename we choose to implement. The fancy version of Kodename will allow the user to have access to various control-flow statements such as if-statements and loops. These Code Blocks will be available when creating a Kodename program as well as when the user creates a Custom Code Block. By introducing the availability of control flow statements and Custom Code Blocks, the complexity of a valid Kodename program greatly increases and will require a more extensive "compiler" to ensure a given program or Custom Code Block is error-free. A successful implementation of this fancy feature hinges less on a clever compiler algorithm and more on a well designed model. As long as our Code Block classes are created in an efficient way to allow for easy identification, it should be feasible to conduct a linear read through of a program and determine if it will generate syntax errors or not.

By partitioning Kodename into three different versions, such that each with their own set of features and levels of difficulty, we believe that we will have an effective implementation strategy that will allow us to draw up the architecture for both a feasible and feature-rich application that accomplishes our primary goal of teaching a younger crowd the techniques and paradigms of computer programming.

System Breakdown

Our incremental implementation of this project will be to ensure that the essential features get completed by the end of the semester. This division will be done in three parts: kernel, normal, and fancy. The Kernel system will consist of the essential features that are required and will be completed within the time frame allotted. The Normal system sees an overall increase in the system features as well as greater attention to user interactions with the program. The fancy system is what we intend to complete, but time may prevent us from doing so. The fancy system is a culmination of all of the features that our team could think of that would maximize the user's experience with our program, relying on features that teach users programming skills, while providing them with an easy to use interface. Each system sees an increase in functionality, user interface, and efficiency.

Kernel System

- **Pre-made Worlds**
- **Single Sessions**
- **Sequential Programming**

The Kernel System has an emphasis on the necessary features that must be included in the program, so a lack of emphasis is put on non-essential features and user interface. When the user begins Kodename, they will be given the option of a number of pre-made worlds that include beepers, walls, and other objects. In the Kernel System, users cannot create their own worlds. This system includes single user sessions, disallowing saving or loading any macros or programs a user creates. There are no help interfaces, undo operations, macro creation, or clearing operations in the Kernel System. Code is added by double clicking the desired action and it will appear in the code panel. Users can only code sequentially in this system, meaning that they cannot place an action in the middle of the program. All actions created will be appended to the end. Editing of nested code statements can only be done thru the whole code block. This means that deletion of a statement in an if statement requires the deletion of the entire if block. Program-tracing is done over an automated play through. The executed

program will play out in real time. The console will displays error-related messages and when a user succeeds/fails to “solve” the world.

Normal System

- **Randomly Generated Worlds**
- **Save/Load Work**
- **Program Tracing**
- **Macro Creation**
- **Single Action Undo**

In the Normal System, more features are added and existing features are improved upon. In a step up from the Kernel System, the Normal System allows for randomly generated worlds. The user has the added ability to select a world at random, or they can hand pick a world that they would like to program in. The Normal System gives users the option to save their progress and come back to it at a later time. A useful guide in the form of tool tips information for functions will teach novice users of our program how to go about solving worlds. The quick ability to undo the last action performed can be done with the undo button. The clear operation is fully supported. A user can remove all code with a button click to begin fresh. Karel source code is fully supported including code tracing that allows users to compare the actions that play out in the world with the Karel source code produced and executed line by line. Program tracing provides the additional option to manually trace code at the user’s speed to further understand coding. Macros or user created functions can be used in the program. Editing of elements that are nested are allowed; an improvement over editing of an entire block. The console displays all program status messages (“Karel moved one space”). A user can simply drag and drop actions from the action panel to the code panel. They can edit code anywhere in the program and are not limited to only adding code to the end.

Fancy System

- **Create A World**
- **Auto-Save**
- **Multi-Action Undo**
- **Code Tracing Delay Timer**
- **Real-time Compiler**

The Fancy System includes features that would be ideal to implement if time were not an issue. The Fancy System includes upgrades that are above and beyond what is required for this project. World editing will be available in this system. A separate page will give users tools and constructs that can be placed anywhere in their world. This world can be saved for further usage. An auto-save feature will be in this Fancy System that will no longer require users to manually save their progress. A step-by-step tutorial of program functionality and usage will teach users what each function does and how they can be used. Users can undo multiple actions up until the first action they performed during the program’s session. Users can also undo a clear operation if they no longer wish to start over. A delay timer for code tracement playing is added to the Fancy System that will allow users to look in depth at what each action does to the world. The console will act as a real-time compiler that detects and reports to the user whether an infinite loop will occur, a wall is attempted to be moved thru, or the character will exceed the bounds of the world.

Glossary

Code Block: a piece of code, e.g. boolean statement, control flow statements, general functions
Code View: the region where the user will add, edit, and delete Code Blocks

Conditionals: Stipulations that must be met in order for some other actions to execute. These include the if and while statements, which will execute the block of code following it if the expression in parentheses is evaluated to true.

Console: the region where any messages from Kodename, e.g. errors, hints, and statuses, are displayed

Custom Code Block: a piece of code that is user created using default Code Blocks

Kodename: the name of our program

Main Window: the primary window where a Karel program is constructed and played

Prompt: A display screen indicating that application is waiting for a decision from the user and some case, the prompt requires input from the user.

Setup Wizard: the wizard that allows the user to either randomly generate a World or create one themselves

Syntax: a general set of rules for the spelling, grammar and structure of a programming language.

User Interface: the means by which the user interacts with the software or hardware of a computer system in a natural and intuitive way.

World: the maze-like environment that is either randomly generated by Kodename or created by the user using a sandbox-like map editor

Summary

Kodename is an educational program for teaching programming to children. It will produce Karel code that can be inspected, if desired. However, all programming will be done through a graphical user interface that uses the drag-and-drop paradigm. In addition, easily readable pseudocode will be visible by default. In this way, lexical and syntactic errors are avoided, and users can focus on the logic behind a program.

The use of the Karel programming language also affords the ability to construct different Worlds that can be used to provide different tasks, goals, or lessons. Instructors will be able to design, save, and distribute Worlds to their students. We provide no networking capability, but platform-independent save files can be hosted on web servers, sent via email, etc.

Kodename will be written in the Java programming language and there will be some cross-platform compatibility. It will run on Windows, OS X, and Linux. It **may** run on other systems that possess a full Java virtual machine.

Acknowledgements

Functional Summary……Miracle Okubor
Use Cases……Team
FSM Diagram……Benjamin Salah
BNF Specification……Stephen Chung
Feasibility Analysis……Isaac Tyan
System Breakdown……Joe Ancona
Glossary……Team
Summary……Stephen Chung

Document Editor……Isaac Tyan