Visual Programming Environments as a Tool to Educate University Students With No Programming Experience

Mark Grivainis, University of Cape Town

Games and simulations offer an engaging method of keeping students interested in learning the basics of programming. This paper researches the different visual programming environments categorised by their classification as code based interfaces where users write code in a given language or drag and drop interfaces which do exist for students, with a limited amount aimed at university students. The type of game that we are interested in making would be classified as a Serious Game as its primary goal is to educate students with the entertainment serving as a means to get students interested in the concepts which they must learn. The most appropriate language for starting students was found to be Python as little syntactical knowledge is required to start making working programs.

1. INTRODUCTION

The fundamental elements of programming are often difficult for new students to grasp [Lahtinen et al. 2005]. Considering that all facets of programming build on these fundamentals, it is crucial that students have a good understanding of how they work and how to apply them.

There are a number of visual learning environments that attempt to assist students in learning these concepts. One of the main drawbacks to these environments is that university students might not find them engaging as they are aimed at a younger audience.

This review outlines the work that has been done on various visual learning environments that are currently in use as well as which sections of research my project will utilize in order to provide an improved visual learning environment.

2. RELATED WORK

2.1. Education through Games and game genres

As said by Kowit Rapeepisarn in The Relationship between Game Genres, Learning Techniques and Learning Styles in Educational Computer Games

> in order for the potential of an educational game to be utilized efficiently the educator needs to design or change the material in the game effectively to support the learning being done.

That being said it can be noted that the educational value of a game needs to be evaluated intensively before it can be used advantageously in the classroom. The table below indicates how different game genres have different impacts on the content being taught and the activities which would correlate with said content [Rapeepisarn et al. 2008].

Learning Content	Learning Activities	Possible Game Styles
Facts : laws, policies, product	Questions, memorization, drill, association	Game show competitions, flashcard types game, mnemonics
Skills: interviewing, teaching, management	Imitation, feedback, coaching, continuous practice	Persistent state games, role-play game, detective games
Judgment: management, decisions, timing, ethics	Reviewing cases, asking questions, feedback, coaching	Role-play games, multiplayer interaction, adventure game, strategy game, detective game
Behaviors: supervision, self- control, setting example	Imitation, feedback, coaching, practice	Role-play game
Theories: marketing rationales, how people learn	Logic, experimentation, questioning	Open ended simulation games, building game, construction games
Reasoning: strategic & tactical thinking, quality analysis	Problems, examples	Puzzles
Process: Auditing, strategy creation	System analysis & deconstruction, practice	Strategy games, adventure games
Procedure: assembly, bank teller, legal	Imitation, practice, play	Timed games, reflex games
Creativity: invention, product design	play	Puzzles, invention games
Language: acronyms, foreign language	Imitation, continuous practice, immersion	Role-play games, reflex games, flashcard games
Systems: health care, markets, refineries	Understanding principles, graduated tasks	Simulation games
Observation: moods, morale, inefficiencies, problems	Observing, feedback	Concentration games, adventure games
Communication: appropriate language, involvement	Imitation, practice	Role-play games, reflex games

Fig. 1. Code before preprocessing.

Considering that programming as content most appropriately falls within the categories of Reasoning, Process and Creativity, we believe that, using Prensky's method, the best possible genres to investigate would be strategy or puzzle.

2.2. Serious Games

Serious games are games in which education is portrayed as a primary ingredient of the games content [Susi et al. 2007]. The idea of using games for purposes other than entertainment was formulated in the book Serious Games by Clark C. Abt(1975). He states

> ”We are concerned with serious games in a sense that these games have an explicit and carefully thought-out educational purpose and are not intended to be played primarily for amusement.” (Abt 1975, p.9).

There are around three generally accepted classifications in which education can be relayed through entertainment, Linear positive(facilitator hypothesis), linear negative(distraction hypothesis) or inverse U-shaped(moderate entertainment hypothesis) . In Linear Positive relationships, the more entertaining a game is the more effective the learning will be. Entertainment distracts the user from learning in Linear negative relationships and a balance is needed to keep the game entertaining and educational. Inverse U-shaped relationships imply that past a certain point entertainment starts to impair the users learning outcomes [Breuer and Bente 2010].

We feel that our game can be classified as Linear positive, this is due to the fact that the user will only be able to interact with our virtual world by using the methods that are to be taught. The more entertaining our game is, the more time the user will spend interacting with it and improving their understanding of the course content.

Using the table for categorising serious games as described in [Breuer and Bente 2010] the game we wish to develop for education can be categorised as follows:

Table I. Label/tag categories for classifying serious games

Label/Tag Category	**Exemplary Labels**
Platform	Personal Computer
Subject Matter	Computer Science
Learning goals	Basic programming language constructs, algorithmic problem solving
Learning principles	Exploration, trial and error, Observational learning
Target audience	First year University students with little to no programming experience
Interaction mode	Single Player, Multiplayer, Co-Tutoring
Application area	Academic Education
Controls Interface	Mouse and Keyboard
Common Gaming labels	Puzzle, Strategy, Simulator

This table allows us to to define all the aspects which the game must encompass while remaining extremely flexible and open to any changes that may need to be enforced at any stage during the development of the game.

3. VISUAL PROGRAMMING ENVIRONMENTS

3.1. Drag and drop interactions

A drag and drop interface is one in which the programmer selects an element, be it a variable or function and drags it into place within a visual environment of sorts. This task is repeated for each variable and/or function required and once all variables etc.

are placed in the correct manor, running the program will yield the required result [Guibert et al. 2004].

3.1.1. Scratch. :
Scratch is designed to be used as an after-school educational game. It has a young target audience ranging in ages from 8 - 16. The majority of contributors to their online repository are around 12 years old. Even though the software is aimed for a young audience there are a number of adults who also contribute.

There is a single user interface which consists of a command palette, scripting window, sprite selection window and stage which can only represent 2 dimensional objects or sprites.

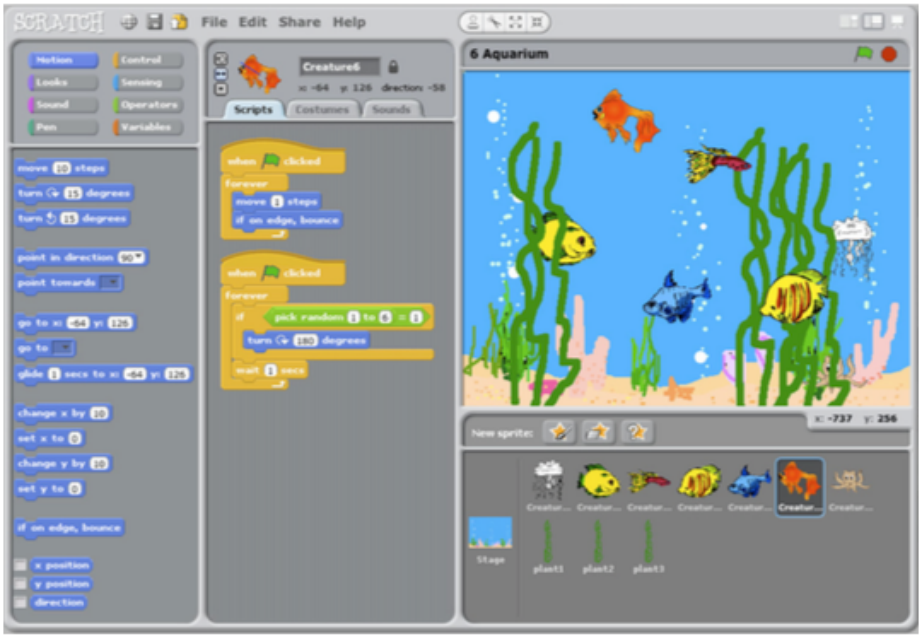

Fig. 2. The Scratch user interface.

All commands are represented as blocks, there are a number of controls structures such as if else statements, repeat loops and user input. These blocks can only fit together in a certain way, this helps prevent errors in the code while allowing the freedom to create animated stories, games and interactive presentations. If fields on the block are incorrectly filled in the block will be highlighted in a red colour. This instantly alerts users that the program will not compile.

During execution the current block that the program is on will be highlighted. This helps users see the effect their block has on the simulation [Resnick et al. 2009] [Maloney et al. 2010] [Maloney et al. 2008] [Fincher et al. 2010].

The fact that adults are utilizing and potentially learning from the interface which scratch provides is a positive notion which allows us to learn from their design with respect to certain aspects going forward. User freedom whilst being notified to errors at an early stage (via the red blocks) is a feature which could prove helpful in our final design.

3.1.2. Alice 2. :
While Alice and Scratch are both block-based environments, Alice supports a 3D stage and library of 3D objects. As much as Scratch is utilized by older audiences, Alice is aimed at a slightly older audience(around ages 12-19). The use of Alice in first year computer science has had great success with students who had no previous experience in programming. Students with no programming experience who were taught using Alice had grades comparable to students who had programming experience and a large percentage of these students continued with second year Computer Science [Dann et al. 2012].

Alice is based around Java, and provides a platform for students to progress into using regular Java. The commands are represented as blocks which take a number of values to perform a function. This system allows students to avoid many syntax errors and create a learning environment that is more about learning concepts than syntax. Alice includes an interpreter and visual object based language that can be used to teach students arrays, loops, functions and variable passing. The basics of parallelism are also taught by allowing students to choose whether a block should be run sequentially or concurrently. When a user runs an Alice program, all assets that will be used must be loaded at the start. This results in long load times for complicated scenes. Its is not possible to make dynamic objects during runtime [Villaverde et al. 2009].

Overall drag and drop interfaces show promise to a younger as well as an older audience, however learning concepts of programming may be well taught via this forum, actually grasping the more abstract aspects of programming once transferring to a code based platform may prove challenging. As such we have decided to further explore current code-based platforms.

3.2. Code based interactions

Code based visual learning environments use regular syntax to interact with the stage. This syntax can be using a specific language or a subset of a language.

Below are examples of some of the well known existing code based learning platforms.

3.2.1. Logo. :
Logo was designed in 1967, it is mainly remembered today for its use of Turtle Graphics in which a user can draw lines by giving a turtle commands to move around the canvas. While it has been used to teach basic programming logic it is more suited to teaching mathematical thinking. There is a python library that gives all the functionality of the language in a more modern setting [Agalianos et al. 2001].

3.2.2. Alice 3. :
Alice 3 offers all of the same functionality that Alice 2 offers with the added ability to export the project to an IDE and then use text-based java instead of the drag and drop commands [Dann et al. 2012].

This allows for all the advantages of Alice 2 whilst potentially affording a more natural transition into programming for real-world solutions.

3.2.3. Greenfoot. :
Greenfoot is a Java based visual programming environment. It is aimed at high school students with some experience in programming. It has a built in editor compiler and debugger. Greenfoot contains features such as an object inspector, popup menus for objects, object state monitoring (watch list essentially) [Henriksen and Kölling 2004]. The key advantages of Greenfoot as a platform could be noted as flexibility, expandability and social interaction [Kölling 2010].

That being said, Greenfoot is not aimed at beginner programmers and as such is not entirely useful for our cause. However monitoring object states as well as a debugger could be important features which would be a noteworthy feature to keep in mind for our final design.

3.2.4. Jeroo. :
Jeroo as a platform is used predominantly at University, the objective is to make a kangaroo collect flowers and avoid nets. It has no explicit variables or data types, however the syntax is close to that of java and c++. The interface contains the level creator and code window as the main means of interaction. A distinguishing feature would be it's containing of the directional constants (LEFT, RIGHT, AHEAD, HERE, NORTH, EAST, SOUTH, WEST). Methods such as isFlower(), isWater(), isJeroo() etc. are key functions which are interacted with by the user and contains three control structures, namely while loops, if and if-else statements, boolean operators AND, OR, NOT [Sanders and Dorn 2003].

After investigation, Jeroo is very limited in what it can teach, however we can take from the fact that in order to teach programming,the language needs to be similar to existing or target languages. That being said, Jeroo is limited to the constructs which it can teach and as such could be improved upon.

4. PYTHON AS A LANGUAGE FOR NEW STUDENTS

Python was originally intended to be an educational language until it was further developed by practical programmers. There is clear evidence that students enjoy python as a first language as it does not carry the nuances of a commercial language [Radenski 2006]. Python does not over reach on what students can do and instead places focus on things such as variable manipulation and function use. Java has complex syntax, even for the hello world program. New students should have to worry less about syntax and more about the code [Miller and Ranum 2005].

As an example we can use the hello world example for both programs.

```
// Java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

```
// Python
print "Hello World"
```

It is clearly visible that python requires far less syntactical consideration than Java in this instance.

Another consideration when choosing a language for learning is whether or not you wish to teach object orientation from the get go. Considering that this is not within our scope, Python seems to check all the boxes required.

5. VISUALIZING SYNTAX THROUGH FLOWCHARTS

Raptor is a learning aid which is flow chart based in that the program is written entirely by dragging elements from a palette into a flowchart structure which will then run the elements as they appear in the structure. This appears bizarre, however in doing so enforces structure and flow principals from an early stage in the learning process [Carlisle et al. 2005]. As a primary learning means Flowcharts offer very little

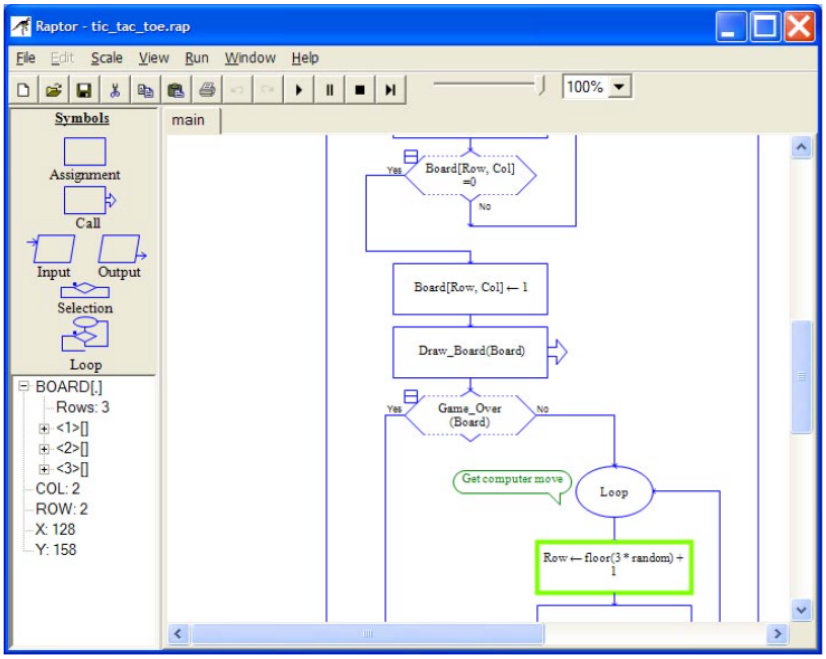

Fig. 3. Raptor in action.

advantage over traditional methods (elaborate) [Shneiderman et al. 1977], this being said there may be merit in them as a compound learning aid. It could be argued that too much time is spent teaching syntax when it could be advantageous to get a better feeling for how code flows [Crews Jr and Ziegler 1998]. However we still believe that flow charts and similar constructs are more effective as attributing tools whereby the user can simply visualize the work he/she has already put into action by means of a flow chart of sorts.

6. CONCLUSIONS

Based on the research conducted, it can be noted that both code-based and drag and drop interfaces have their strengths as well as weaknesses. As the students are expected to know how to write code that works by the end of the course we feel that, while drag and drop interactions may be useful in the start, they may hamper the students moving forward. On the other hand a code based interaction, although maintaining a slightly longer learning curve seems to remain beneficial to the students in the long run and as such we have decided to utilize this approach in the final design. We also noticed that, while flowcharts may be an effective peripheral, will not be effective as a main focal point of the interface.

Students seem to react well to single window interfaces as is evident in Scatch, Alice and Jeroo. As such we have deemed it appropriate to take that into account with our design.

Finally we believe that, considering the aim of the project, Pensky's design consideration table as well as the categorisation of serious games our game will fall within the following category:

A puzzle, strategy simulator used to teach basic Computer Science constructs, released on a desktop platform with a code based interface, interacted with via keyboard and mouse, targeted at University students with little to no experience in programming.

REFERENCES

Angelos Agalianos, Richard Noss, and Geoff Whitty. 2001. Logo in mainstream schools: the struggle over the soul of an educational innovation. *British Journal of Sociology of Education* 22, 4 (2001), 479–500.

Johannes S Breuer and Gary Bente. 2010. Why so serious? On the relation of serious games and learning. *Eludamos. Journal for Computer Game Culture* 4, 1 (2010), 7–24.

Martin C Carlisle, Terry A Wilson, Jeffrey W Humphries, and Steven M Hadfield. 2005. RAPTOR: a visual programming environment for teaching algorithmic problem solving. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 176–180.

T Crews Jr and Uta Ziegler. 1998. The flowchart interpreter for introductory programming courses. In *Frontiers in Education Conference, 1998. FIE'98. 28th Annual*, Vol. 1. IEEE, 307–312.

Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated transfer: Alice 3 to java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 141–146.

Sally Fincher, Stephen Cooper, Michael Kölling, and John Maloney. 2010. Comparing alice, greenfoot & scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 192–193.

Nicolas Guibert, Patrick Girard, and Laurent Guittet. 2004. Example-based programming: a pertinent visual approach for learning to program. In *Proceedings of the working conference on Advanced visual interfaces*. ACM, 358–361.

Poul Henriksen and Michael Kölling. 2004. Greenfoot: combining object visualisation with interaction. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. ACM, 73–82.

Michael Kölling. 2010. The greenfoot programming environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 14.

Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin*, Vol. 37. ACM, 14–18.

John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 16.

John H Maloney, Kylie Peppler, Yasmin Kafai, Mitchel Resnick, and Natalie Rusk. 2008. Programming by choice: urban youth learning programming with scratch. *ACM SIGCSE Bulletin* 40, 1 (2008), 367–371.

Bradley N Miller and David L Ranum. 2005. Teaching an introductory computer science sequence with Python. In *Proceedings of the 38th Midwest Instructional and Computing Symposium, Eau Claire, Wisconsin, USA*.

Atanas Radenski. 2006. Python First: A lab-based digital introduction to computer science. In *ACM SIGCSE Bulletin*, Vol. 38. ACM, 197–201.

Kowit Rapeepisarn, Kok Wai Wong, Chun Che Fung, and Myint Swe Khine. 2008. The relationship between game genres, learning techniques and learning styles in educational computer games. In *Technologies for E-Learning and Digital Entertainment*. Springer, 497–508.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and others. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (2009), 60–67.

Dean Sanders and Brian Dorn. 2003. Jeroo: a tool for introducing object-oriented programming. In *ACM SIGCSE Bulletin*, Vol. 35. ACM, 201–204.

Ben Shneiderman, Richard Mayer, Don McKay, and Peter Heller. 1977. Experimental investigations of the utility of detailed flowcharts in programming. *Commun. ACM* 20, 6 (1977), 373–381.

Tarja Susi, Mikael Johannesson, and Per Backlund. 2007. Serious games: An overview. (2007).

Karen Villaverde, Clint Jeffery, and Inna Pivkina. 2009. Cheshire: Towards an Alice based game development tool. In *International Conference on Computer Games, Multimedia & Allied Technology*. 321–328.