Balanced Game Design Approach on Entry-Level Programming Education for Adults

Mohammad Hassan Salim
Georgia Institute of Technology
VA, USA
msalim7@gatech.edu

ABSTRACT

Technology offers several methods of teaching programming. One way is the use of educational games. Educational games offer a more exciting way to learn. There have been numerous game design theories applied onto education. Balanced Game Design (BGD) is one of these theories. In this paper, I present an educational game I built using BGD that will teach entry-level programming to adults. I will breakdown how I applied the BGD theory into code such that these steps could be used for any subject matter. Additionally, I will discuss how well the game fits the BGD model. Finally, I will explore how well the game teaches programming thus testing how well BGD works.

Author Keywords

Educational game; Serious game; Balanced game design; Evidence-centric design; Game-based learning.

INTRODUCTION

Educators have used various forms of game-based learning to teach students. To justify developing educational games, which is a specific type of game-based learning, I must prove that educational games work. I will explore the efficacy of educational games by investigating two studies.

Professors at Clemson University conducted a congregated analysis of other studies on the impact of different categories of game-based learning [1]. There is one caveat: all the game studies contain games from 2010-2014. It was difficult to find an analysis on a large dataset of games developed after 2015. The study has four main categories of games: educational/serious, entertainment (purely entertainment game used for the used for the purposes of teaching), design-based tools (such as Scratch), and mobile (augmented reality).

Although design-based games were considered more effective than educational games (see Table 1), design-based games usually needed an instructor to assist using it. The study mentions that educational games were still very effective in the cases where the education and entertainment aspects were blended together well.

Professors from the City University of New York conducted an experiment using serious games in college classes [2]. The class subjects were English, math, and science. The professors would have two sections for each class. One class would have traditional learning and the other would game-based learning. After each lesson, both sections would receive a quiz to see how well they retained their knowledge. The grades are very close to each other for both game-based and traditional courses for most cases (see Table 2). Game-based learning was generally better for surface learning and traditional learning was generally better for deep learning. Math professor 3 and science professor 3 have major differences in Overall Learning. This could be due the professors not utilizing the games well, have an unusual grading scheme, or content that is too deeply difficult to be taught via game mechanics.

Game Genre	*Number of Papers (total=29)*	*Number of Results (total=97)*	*Large Effect Size (total=24)*	*Moderate Effect Size (total=9)*	*Small Effect Size (total=3)*
Educational (or Serious)	11	40	7	5	2
Entertainment	4	17	4	0	0
Designed Based	8	28	13	4	1
Mobile (AR)	6	12	0	0	0

Table 1. Number of papers per game-based learning category with effect categorized by large, moderate, or small sizes. Remaining papers did not have an impactful effect size.

CONF '22, Jan 1 - Dec 31 2022, Authorberg.

unique doi string will go here

Professor	Surface Learning	Deep Learning	Overall Learning
English Prof. 1 (game)	2.627*	2.490*	
English Prof. 1 (non-game)	1.582	2.275*	
English Prof. 3 (game)	62.5%	52.1%	
English Prof. 3 (non-game)	62.5%	63.9%	
Math Prof. 1 (game)	85.2%	55.9%	73.5%
Math Prof. 1 (non-game)	79.9%	63.6%	73.7%
Math Prof. 2 (game)	92.5%	82.5%	89.6%
Math Prof. 2 (non-game)	95.3%	80.2%	89.9%
Math Prof. 3 (game 1)			50.0%
Math Prof. 3 (non-game 1)			49.3%
Math Prof. 3 (game 2)			50.5%
Math Prof. 3 (non-game 2)			65.6%
Science Prof. 1 (game)	79.4%	67.5%	
Science Prof. 1 (non-game)	75.5%	77.1%	
Science Prof. 3 (game)			61.4%
Science Prof. 3 (non-game)			44.6%

Table 2. Class means of post-lesson quiz grades. For a given professor, the same quiz was given and graded in the same way for both game and non-game classes.

***For the first English professor, the letter grades were converted to grade points.**

However, students notably enjoyed the game-based learning more (see Table 3). This is very useful since we know adults can learn roughly the same as in a classroom but enjoy it far more if done with game-based learning.

Existing Educational Games

It's evidential that educational games can be effective and enjoyable at teaching earlier college levels of math and science if the educational and entertainment aspects are fused together correctly. It's also evidential that there are existing successful educational games. Since the goal of this research is to teach programming through educational games, I will trim the scope down and analyze some highly critical and well-reviewed educational games on programming.

After doing some research, it seems most successful serious games have a target audience for children. These games include CodeCombat and CodeMonkey (see Figure 1).

	English	*Math*	*Science*
Non-Game	84.1%	61.3%	91.3%
Game	96.4%	76.5%	100%

Table 3. Enjoyment of the lesson on a Strong Disagree to Strongly Agree scale. Agree and Strongly Agree were considered as enjoyable.

Figure 1. CodeMonkey is a game where player enters scripting commands to have a monkey move through obstacles and collect items.

After some more digging, I was able to find adult-level programming games. These include Screeps, HackNet (see Figure 2), and Else Heart.Break().

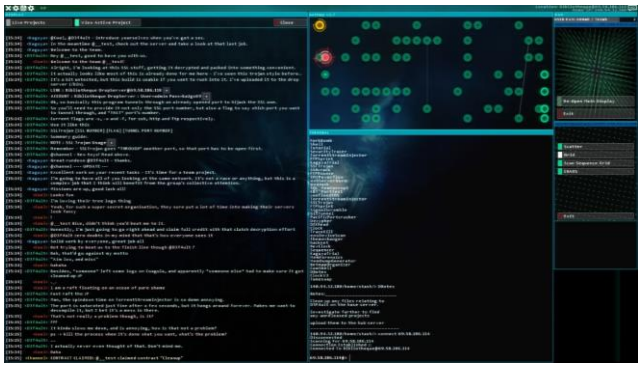

Figure 2. HackNet is a console-programming game driven primarily by programming and story.

These games were well reviewed, but they all share one attribute: they require programming experience. The target audience is for adults who enjoy programming already. I could not find any educational games for newcomer programmers with an adult target audience. Hence, this will become my goal. I will attempt to bridge the gap between the educational games and post-graduate adult education and computer science. This will contribute to the educational game community as well as the computer science community.

Use Case

I will now set context and establish motivation for making a entry-level programming educational game targeted towards adults. Educational technology has a wide target audience due to engaging learners who have difficulties attending a conventional classroom [3]. These difficulties include:

- Geographically dispersed with limited time and resources.
- Living far from education centers such as universities.
- Busy working during school hours.
- Family commitments.
- Located in conflict or post conflict area.
- Disability or illness preventing relocation to school.
- Cultural or religious beliefs or holidays schedules do not compliment well with the class schedules.
- Communication difficulties due to foreign languages or shy learners.

Notice that this is a list of difficulties primarily applies to adults therefore adults should be at least one of the target audiences for educational technology. My focus is on educational games for learning programming at a newcomer-level catered for an adult audience. Why is this? Imagine choosing (or being forced into) a career path that is not very rewarding or fiscally gainful. One option is to change careers. As an adult, it's sometimes more difficult to learn via MOOCs even if they solve some of the inconveniences listed above. The last thing an adult wants to do is homework in their free time. Adults usually want to watch Netflix or play a video game. Therefore, this is my use case. For the remainder of the paper, I will explain my research goals, discuss my development execution, and analyze testing results.

RESEARCH GOALS

Before discussing the development details, I want to present two theories I used to help guide me in developing an effective and enjoyable educational game. The first theory is Balanced Game Design. Additionally, it is the primary theory for my research goals. I'm using the game I develop to essentially test the effectiveness of the BGD theory. The second theory is a I-Feature. I am using only a part of the I-Feature theory to fill in the gaps that BGD doesn't really cover in order to make the game more enjoyable.

Balanced Game Design (BGD)

MIT's Better Learning in Games [5] defines learning as two interrelated questions: what are you trying to teach and how do you know they know it? The first part is content, and the second part is assessment (see Figure 3). Educational games often do not apply assessment-based game design. It is important to know if the student learned the content after presenting the different parts. Assessment-based game design is the root of the BGD.

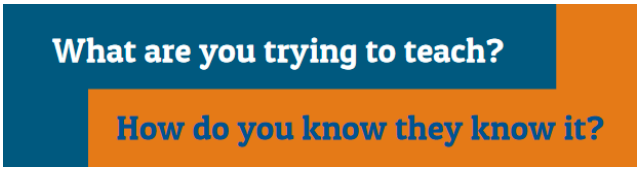

Figure 3. Interrelated questions to consider before designing an educational game.

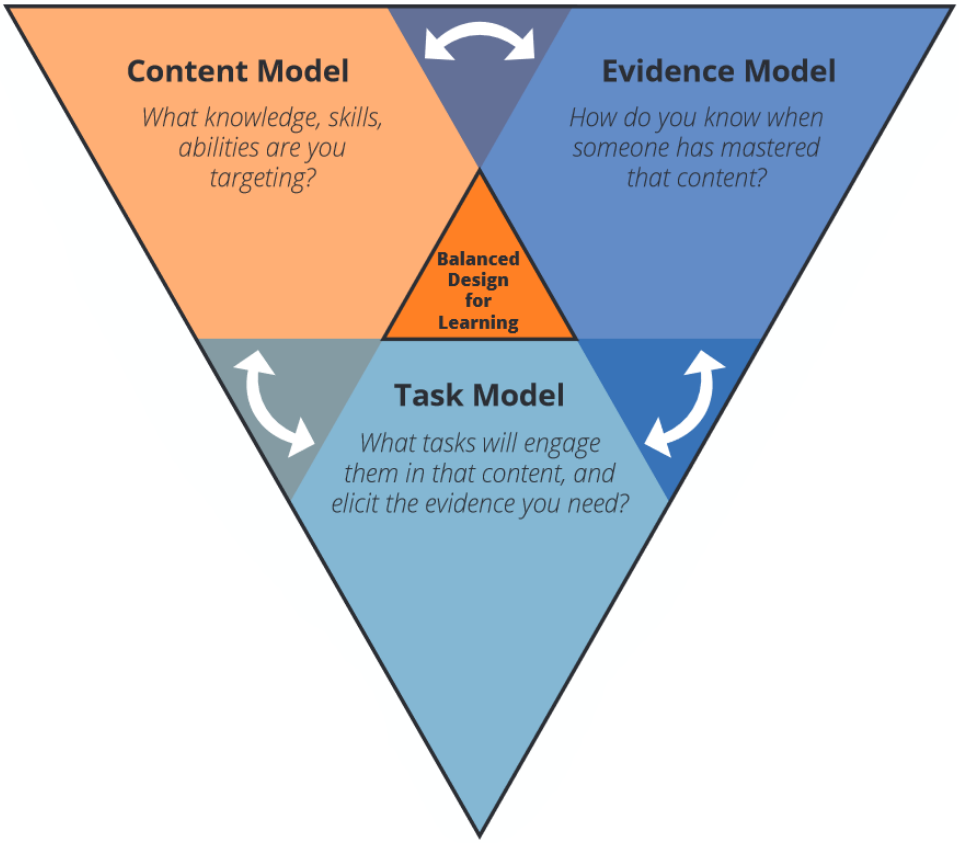

Figure 4. BGD uses Evidence-Centered Design (ECD) to build a three-part circularly connected design with the following models: content, task, evidence.

BGD a learning design approach that should be integrated with whichever game design approach you prefer such that it better aligns learning goals with the game mechanics. This not only will help drive engagement (which is a key goal of I-Features), but also make the educational game more effective. BGD builds off the Evidence-Centered Design or ECD (see Figure 4).

ECD focuses on the measurable success of a student's learning. Balanced Design has three models: content, task, and evidence (see Table 4).

Model	Definition	Example
Content	What skills we teach the student.	Entry-level programming for adults.
Task	Which goals does the student need to complete.	Playing a video game.
Evidence	Knowing if a student has mastered the content.	Completing the game or how well the student has completed the game.

Table 4. Content, Task, and Evidence definitions and example.

The Content/Task/Evidence models seem intuitive to develop with and measure results, so I will be using BGD as my primary guideline in development. In addition, I will be testing how well BGD works is in practice.

I-Feature

I-Feature [4] theory is a set of key features in educational games that drive engagement which also happen to start with the letter "I". These include identity, immersion, interactivity, increasing complexity, informed teaching, and instructional (see Table 5).

I want to have a focus on Immersion from the I-Feature theory because that's the most important feature that isn't really covered by BGD. This can be as simple as creating enjoyable dialogue or story. BGD focuses on creating an effective learning experience without any framework for creating an enjoyable learning experience. Since BGD can be easily integrated with other frameworks, it would be beneficial for me to take advantage of the I-Feature's Immersion guidelines and create an entertaining educational game.

DEVELOPMENT

I will discuss the limitations that I had which effected the development of the educational game. After that, I will

discuss the steps I used to apply the BGD into my code. These steps will be abstracted out such that it can be applied to other game genres. The source code for my game can be found at https://github.com/mohsalim/EdTechGame.

Feature	Example
Identity	Customizable or relatable hero
Immersion	Fantasy universe that smoothly intermingles game play mechanics and educational goals.
Interactivity	Responsive UI with interesting non-playable characters (NPC).
Increasing Complexity	Each new level or mission introduces a new programming element such as loops or recursion.
Informed Teaching	NPC communicate programming lessons in the form of tutorials or instructions per challenge.
Instructional	NPC communicate the strictness of programming as the rules of the universe.

Table 5: Example of each I-Feature.

Limitations

Due to short development time and little game development experience, I picked Unity as my game development framework as it's easy to develop a game alone with Unity. I programmed in C# and compiled with .Net 4.6.1 (experimental for Unity). Initially I wanted to teach C#. However, Unity uses Mono (or MonoDevelop) under the hood which has issues running C# code on the fly after building the executable file. It worked fine during in-editor run. I had to switch to Python. I used IronPythong to run Python 2.7 code. I based my game's lessons on an existing Python tutorial https://learnpython.org.

Moreover, due to short time and lack of game development experience, I took notes from HackNet. I went with a console-like game where the story drives engagement. However, HackNet has a more serious tone. I opted for a casual and comical tone, so it would provide a welcoming atmosphere.

BGD to Code Overview

There are five steps to apply BGD to your code:

1. Create BGD document for each lesson you are trying to teach.
2. Map each row to a certain Level or Scene with Dialogue or Story details.
3. For each row in the BGD document, extend from abstract Problem class that contains the problem contents and validation.
4. Organize the extended Problems into Tutorial/Level sets.
5. Level Manager class will contain a Tutorial/Level Queue and will dequeue as the player progresses the game.

Step 1: BGD Document

Step one is very important. It can be very time consuming, but it will save a lot of design headaches down the road. It has three steps:

1. Create a list of skills you want the player to learn and place this in the Content column.
2. For each Content item, create a Task that needs to be completed.
3. For each Content/Task pair, create an Evidence item. It will contain what it means if the player does this Task correctly and list of possible explanations if the player does this Task incorrectly.

The Content model contain lessons such as outputting to console. The Task model will contain the technical details of the game. In this case, it's a simple Hello World program (see Table 6, C1). The Evidence model is the real meat of this document. It will articulate what it means to be correct. Usually this is surface learning of the Content. It will also list out the possibly incorrect cases. For each incorrect case, it provides details which will be used for as a Hint to the user. This Hint can be delivered through Dialogue from an NPC, a modal box, etc. Some lessons reuse previously taught

content (see Table 6, C2). This is a good opportunity to see if the player has deeply learned a lesson.

Step 2: Story Document

For each Tutorial or Level, there will be a story to motivate the player to complete the lessons. Each Level will map to a subset of the BGD document (see Table 7). If your game does not have a story, it is still useful to map levels to lessons. This is an important step if you want to emphasize on immersion and player engagement.

Content	Task	Evidence
C1: Outputting to Console	Observe "Hello World" starting code. Then have it to say print right after: "Goodbye World".	**Correct**: Player learned to write two lines of text to the console. **Incorrect**: 1. ***No change within quotes.*** Player does not understand where the actual text is. 2. ***Only one line printed out.*** Player replaced original string with new string instead of printing one after the other. 3. ***Text has incorrect casing.*** Player did not understand the strict nature of programming (such as case-sensitive strings).
C2: Variables	Four variables will appear on screen: an integer, a double, a string, and a Boolean. Print the four variables one line after the other. Create four more variables (one of each type with a specific value), then print those.	**Correct**: Player learned how to initialize and print four different primitive variable types. **Incorrect**: 1. ***Variables did not all print.*** Failed deeper learning of C1. 2. ***Incorrect values for the four variables created.*** Player does not understand the typing requirement when declaring variables. 3. ***Incorrect number of prints.*** Player has printed too many or too little items.
…	…	**…**

Table 6: BGD document example. Entire document not shown due to large size.

Level	Story	Lessons
L1	Your classmate, Broseph the Brogrammer, needs help on programming homework.	C1, C2
L2	Professor Gevabad Graids hands out a difficult assignment. He expects a printed out copy of the assignment. Broseph insists to drop it off for you.	C3, C4
L3	You receive a 0. Broseph received a 100%. Broseph says he doesn't remember offering to drop off your assignment for you. TA Ova Wurkt offers an extra credit assignment.	C5
L4	You get a message from @n0n_h@ck3r asking if you want a chance to get revenge on Broseph. However, you are framed for the hacker's crimes.	C6, C7
…	…	…

Table 7. Story document maps levels to lessons.

Step 3: Abstract Problem

The abstract Problem class contains three properties and two methods (see Figure 5). It will contain starting code, answer, and instructions as properties. These three make up the Task model. Additionally, there is a method to validate the player's code, output of that code, and hint out-variable that will be updated if the answer is incorrect. This makes up the Evidence model. Finally, there is a method to produce the dialogue. This will contain text, sprites, names, etc. For each row in the BGD document, you want to create a child class of Problem.

If you are creating a non-programming educational game, you can abstract this class more by using state, action, and result like objects instead of strings (see Figure 6).

Step 4: Tutorial/Level Sets

This step serves to organize the Problems into Tutorials or Levels. If you are planning to treat each Problem as its own Tutorial or Level, then you can skip this step. The Tutorial class will just be a list-like object that contains the Problems. The Tutorial class is ideal for optimizing the memory and speed for Problem declaration.

Step 5: Level Manager

The Level Manager will contain a Queue of Tutorials or Levels in the correct order. The Level Manager should be handled based on the framework you are using. For example, Unity will use a MonoBehavior class to play the next Scene or manually update the current Scene.

ANALYSIS

Before I can test my game with players, I must (informally) prove that my game fits the BGD model. If the game does not fit the BGD model, then it is not fairly testing the BGD theory.

The Content model asks, "What knowledge, skills, abilities are you targeting?". The game should be teaching entry-level programming. Each level should be teaching specific lessons of programming. Each lesson is equivalent to a single specific programming technique. For example, the first lesson is outputting using the print function in Python. Before a compound lesson is taught, the individual lessons must be taught first. For example, a lesson that asks to build a function that prints from 0 to 100 would require a lesson that teaches printing and a lesson that teaches for loops prior. This way the content is fairly delivered to the player and their skills will gradually increase per lesson.

The Task model asks, "What tasks will engage them in that content, and elicit the evidence you need?". The BGD document contains the programming tasks, but this only elicits the evidence I need. To engage the player in Content, I tie map Tasks to a story. The story creates a basic sense of immersion using casual and quirky humor.

```
public abstract class Problem
{
    public abstract string StartCode { get; }

    public abstract string Answer { get; }

    public abstract string Instructions { get; }

    public abstract bool ValidateAnswer(string code, string output, out string hint);
    public abstract Dialogue GetDialogue();
}
```

Figure 5. Abstract class Problem.cs for programming education games.

```
public abstract class Problem
{
    public abstract object StartState { get; }

    public abstract object EndState { get; }

    public abstract string Instructions { get; }

    public abstract bool ValidateState(Action[] actions, Result[] results, out string hint);


    public abstract Dialogue GetDialogue();
}
```

Figure 6. Abstract class Problem.cs using states/actions/results for any educational game genre.

The Evidence model asks, “How do you know when someone has mastered that content?”. The Problem class has a validation method. This validates both the code and output. If the correct functions or operators are used with the correct output, then the player has mastered that content.

The combination of the BGD document, story document, and abstract Problem class shapes my game to be a good fit for the BGD theory. Therefore, my game is a fair contender to test the BGD theory.

TESTING

In the testing phase, I observed players beta-test the game. The players had little to no programming experience. I followed up with an interview asking how well they felt they learned, what they recalled, and how much they enjoyed the game. In addition to the interview, the BGD framework tests how much the players learned based on how many tasks they complete. During observation, I made sure to pay attention to anything that would sway the player (positively or negatively). This could be a bug, UX design, game design, or difficulty of the task.

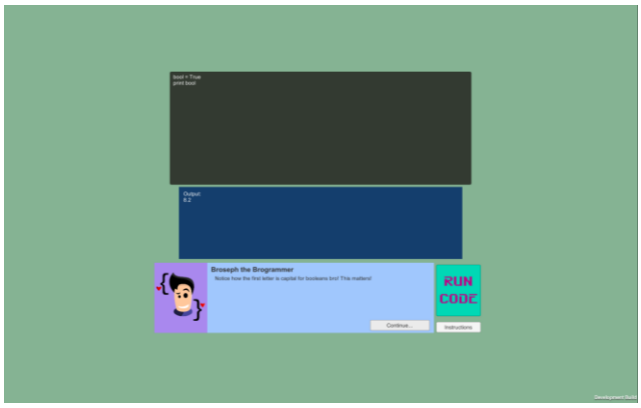

Figure 7. In-game console and dialogue box. Contains output from previous problem.

Players felt as if they learned at the surface level. Players would remember some functions and operators that they covered, but players could not remember the exact syntax. Although some levels tested players deeper understanding by having them reuse functions or techniques learned previously, it would be difficult for the player to recall how to go about it without looking in the detailed instructions section. Furthermore, players felt the game became mentally exhausting in the latter levels. Players enjoyed the simplistic and colorful UX design (see Figure 7). Additionally, players enjoyed the NPC dialogue, names, and story. During my observation, I noticed the lack of variety in game mechanics left the player feeling the game was too long. Players at times felt frustrated if they could not understand why they were failing to get the code compiled. Some minor bugs added to the frustration.

CONCLUSION

In this paper, I developed a programming educational game using BGD. I included an overview of how to apply the BGD in practice. Finally, I tested the game which in turn tests BGD. My test results show that BGD can work. Although there was technical and time limitations, the game was able to still teach some programming concepts at a surface level to adults with little to no programing experience.

Future work can be done. The BGD application steps can be more refined for genre-ambiguous educational games. The game itself can be improved using feedback and analysis. I would like to expand the game to be a top-down world where the player walks to in-game terminals to program. There would be puzzles that would involve programming the terminals and changing parts of the puzzle inside the game world. A better blend between the education and game mechanics would immerse the player more and incentivize better to learn certain programming concepts.

ACKNOWLEDGMENTS

I want to thank professor David Joyner and mentor Jace van Auken for allowing me work on a project that piqued my personal interests while still aligned with the goals of the class. It has been a fulfilling experience. I want to give additional thanks to beta-testers who went out of their way to play my game: Mohammad Ali Salim, Varda Khan, Mohammad Shahvez Haji, Ankush Gupta, and Edgar Tran.

REFERENCES

1. Qian, M., & Clark, K. R. (2016). Game-based Learning and 21st century skills: A review of recent research. Computers in Human Behavior, 63, 50-58. https://www-sciencedirect-com.prx.library.gatech.edu/science/article/pii/S0747563216303491
2. Crocco, F., Offenholley, K., & Hernandez, C. (2016). A proof-of-concept study of game-based learning in higher education. Simulation & Gaming, 47(4), 403-422. http://journals.sagepub.com.prx.library.gatech.edu/doi/pdf/10.1177/1046878116632484
3. Education, A. (2017). based on Course 'Easily Moving from Learning to e-Learning' (Intellectual Output 5). http://www.better-e.eu/images/docs/IO5.pdf
4. dos Santos A.D., Fraternali P. (2016). A Comparison of Methodological Frameworks for Digital Learning Game Design. In: De Gloria A., Veltkamp R. (eds) Games and Learning Alliance. GALA 2015. Lecture Notes in Computer Science, vol 9599. Springer, Cham
5. Groff, Jennifer; Clarke-Midura, Jody; Own, V. Elizabeth; Rosenheck, Louisa; Beall, Michael. (2015). Better Learning in Games. http://education.mit.edu/wp-content/uploads/2015/07/BalancedDesignGuide2015.pdf