Restorative Justice: Introducing Juvenile Hall Students to Computer Science

Erik Mork, Zoe Wood

Abstract: Computer Science is an increasingly important field in regards to (in)equity in the United States and our world. Equal access to adequate Computer Science education is an issue in schools. Incarcerated youth in Juvenile Halls are often unable to obtain classes / resources that would foster interest in the field of engineering. This paper presents findings from a revitalized and revamped computer science course that was originally taught in San Luis Obispo's Juvenile Hall in 2019 [7]. This iteration of the project introduces modified curriculum and different tools for learning the art of programming that was not included in the 2019 class. By adding more structured curriculum such as a scaffolding of base code and worksheets to work on key concepts, this course is more digestible for the Juvenile Hall students and is more repeatable for future Cal Poly students. The data collected shows an increase in students' efficacy in programming from the beginning of the course compared to the end of the course.

1 Introduction:

There is a shocking lack of diversity in the tech industry, particularly in computer science. A lack of diversity is one of the reasons why many technologies we have today reflect and reinforce structural inequalities seen in society [1]. Increasing diversity and ethical thinking in the field must start with education, particularly pre-college education. There are two large areas of improvement needed to increase diversity in engineering. One is recruitment, which is the recruiting of K-12 students into the field of engineering through classes and extracurriculars. Retention focuses on increasing the retention rates of students of color and gender minority students in undergraduate and graduate programs. Recruitment is a larger factor than retention in increasing underrepresented participation in engineering, therefore, high school computer science education is particularly important for recruitment and increasing diversity in the field[2].

Currently, there is a lack of equal access to computer science in high schools in the US. In 2019, 5,820 white students took the APCS test, while only 3,870 Latinx students took the test and 338 Black students. Not only are the numbers of students taking the class unacceptable, but the pass rates are skewed towards white students, which leads us to believe the education given to students, when available, is inadequate for students of color. 39% of white students got a passing grade, 19% of Latinx students got a passing grade, and 21% of Black students got a passing grade [3]. Reasons for both the low rates of students of color taking computer science courses in high school and the lower pass rates are linked with a lack of funding and a biased engineering teaching methodology [4][5]. Educators, sometimes inadvertently, value the cultural capital of privileged groups in society while neglecting the community cultural wealth possessed by many people of color [5].

This paper details our experience redesigning and teaching a computer science course for incarcerated youth in a California juvenile hall. There is a severe lack of access to engineering disciplines in the Juvenile Hall, and this course provides an introduction to computer science with the goal of piquing the students' interests in computer science and furthering recruitment of high school aged students into computer science. The course designed and taught is a game design class that, while teaching a basic understanding of programming and math, also relies on further non-technical skills of art, creativity, and passion that allow for opportunities for students of all interests and skills to thrive.

The course's emphasis on creativity and art encourages students to bring their full selves to their programs and not partition off STEM from other aspects of their lives and their personalities. The emphasis on creativity also works to break down the negative stereotypes often associated with computer scientists of being anti-social, boring, and lacking creativity [6]. Breaking down the stereotypes of computer science opens up the field to a broader spectrum of students and would hopefully lead to an increase of students' sense of belonging to the field.

2 Related Work:

Introducing Computing to a Cohort of Incarcerated Youth:

The work that is most closely related to our project is the Juvenile Hall computer science course that was created and taught by Kirsten Mork in 2019. Our project is a continuation of her work, and a restructuring of her class to make it more shareable and repeatable with other institutions and interested teachers. Kirsten Mork was the first person to introduce the San Luis Obispo Juvenile Hall to computer science when she taught a five week 2-D computer graphics course there. The survey results from the students were overall very promising; there was a noticeable increase in student interest in computer science and in the students' programming efficacy. The work done by Kirsten Mork also validated the use of a 2-D gaming curriculum. The students found the game as being both engaging and conducive to their own creativity [7].

Since all the results of this first intervention were overwhelmingly positive from the students, our work does not stray far from the original class. Instead of changing the topic of the course or the final project, we focused on adding more structure to the course so that the work could be passed on. Mork's curriculum documentation was scattered, but we were able to rework her provided curriculum into a well documented structure that now includes base code, lecture slides, teacher scripts, and student worksheets (both for individual work and group work depending on the class dynamic). While Mork had to input parts of students' code outside of class time due to the shortness of the class as a whole, the addition of base code makes it so that anyone teaching the course will not have to put in extra coding work outside of the class for the students. The lecture slides and scripts are also resources that lower the work needed from teachers outside of class time.

Teaching Introductory Computing at County Jails:

Prior to Kirsten Mork's Juvenile Hall class, Dr. Theresa Anne Migler-VonDollen introduced computing courses to San Luis Obispo county jails. Migler's work is focused on adult populations in jails, while our work focuses on youth populations. This difference in age leads to a difference in curriculum taught and the methods of teaching. Migler's class is a four week class with two ninety minute lectures per week with a weekly two hour lab session. Her class is taught in python and covers the topics of "basic arithmetic, built-in Python functions (e.g. "print", "type"), user input parsing, simple data types, user-defined functions, parameters, simple programs, boolean logic, and loops." [8]. The difference in age also brings in a slight difference in the goals of the course; Migler states that their goals are "to be empowered in logical thinking and mathematical skills, to elevate technical literacy, to be prepared for success at an introductory computer programming course at our local community colleges upon their release" [8]. While the goals of our class generally align with Migler's goals of empowering students and improving technical literacy, however, our goals are also to stimulate the students' creativity and encourage them to bring their creativity to their programs as well as give the students the confidence to explore areas of STEM that are frequently gate kept by negative stereotypes.

Game Programming in CS0: A Scaffolded Approach

This intervention that ran at University of Washington, Bothell, experimented with introducing a CS0 game design course meant to bring CS to non-major students. Similarly to our goal, they hoped to get non CS students interested in pursuing future CS education through an introductory game design course. Their work focused on using a scaffolding method of teaching game design concepts in the drag and drop program of GameMaker and then later recovering the same topic but adding in programming in C#. The emphasis of both game design and programming in their game design CS0 course closely mirrored our approach. We opted towards moving straight into programming after game design discussions instead of using drag and drop programs such as GameMaker. This decision was due to our belief that moving straight into programming would be more engaging for students and due to time constraints of the course we did not have time to practice with other programs. The findings of the Bothell experimental course were positive, with many students providing feedback that showed a preference for eliminating the GameMaker aspect and moving straight into C#.

3 Curriculum:

The course was centered on the design and creation of a 2-D game using the java based programming language, Processing. Our class was a five week course with two 1 hour classes each week. The structure of the class generally consisted of a short 10-20 minute lecture, working together on a selection of problems on a relevant worksheet, and then 30-40 minutes of individual work on games using the tools learned from that day's lesson. Since in the last iteration of the class the teacher had to fill in student's code outside of class due to the short work time, we added a base code into the curriculum to provide a more robust starting point and remove some of the boilerplate code that confused the students in the last iteration of the course.

Lessons:

Lesson 1 - Introduction to Class and Computer Science: Lesson one's focus was on introducing the students to the field of computer science and the goals of this course. We showed the many possibilities there are with programming and showed students the many fields they can work in with computer science. We strongly emphasized the importance of bringing their own backgrounds, passions, and personality to their code both in this class and in any future programming they do. The main goal of this lesson was to emphasize the importance of programming needing to be a creative process and personal process. We then showed the students an example of the type of game they would be creating in the course - the game is a simple 2-D collection type game where the character moves from one end of the screen to the other collecting or avoiding various falling objects. *Lesson 1 Worksheet/Assignment:* The assignment for this class was the same as the worksheet; the student's task was to design a character on paper using simple shapes and labeling the location of each shape in the grid.

Lesson 2 - Shapes and Colors: Lesson two was a longer lesson because this was the first introduction to the language Processing. This lesson we went over the basic commands that we needed throughout the course. We covered how Processing draws the canvas and looked at various basic Processing commands such as: fill, ellipse, rectangle, stroke, and noStroke as well as some others. *Lesson 2 Worksheet/Assignment:* The worksheet for this lesson was a short worksheet that consisted of three different pages of practice for drawing shapes. There was a grid that had different shapes of different colors on it and the students had to write the code commands that would draw the shapes at those locations. The students' individual assignment for the day was to type out all the code for drawing their character from the previous lesson onto the screen.

Lesson 3 - Functions: Lesson three was focused on defining functions and showing the students how they work to package up code and make it more readable and reusable. Since lesson two was a little bit longer and we lost some of the students' attention at the end, we shortened lesson three so the students would remain engaged. We cover what the two built in Processing functions we use, setup and draw, do and how we use them. Then, we described the benefits of functions for organizing code and showed the student's how we would be creating functions for this course. *Lesson 3 Worksheet/Assignment:* The worksheet for this class had two questions that show examples of functions and then asks the students to call whichever function would draw the desired shape on the screen. This was to practice calling functions and get the students used to seeing how functions look and work. The assignment for the day was to try out different pre-made backgrounds by calling each background function in the draw function. After picking out which background function they want the students were supposed to move their character code from the last class into a draw character function and call that new function in draw. The extra time was used to design the falling object for their games and begin coding it.

Lesson 4 - Variables: Lesson four taught students what variables are and how to use them in context of the students' games. The lesson showed what variables are and showed how we could create a variable for the character's x position and use Processing's translate function to move the character across the screen. Then, we did some practice with variable math. *Lesson 4 Worksheet/Assignment:* The worksheet for this lesson mainly focused on how to use the translate function with variables. The assignment for the day was for students to create a variable for the x position of their character and put it in the translate function above their character code, then

increment the x position in draw to make the character move. If there was extra time, they could continue the work designing their falling objects and making them fall from the screen.

Lesson 5 - Conditionals: Lesson five focuses on the function of conditional statements and how we will use them to make the students' character movements better. The lesson shows what conditional statements look like and their function and shows examples of how we would use them in our code. *Lesson 5 Worksheet/Assignment:* The worksheet for this lesson had one example of code with conditional statements in it and the students had to say what the code will do when run. The assignment for the students to complete for their games is to add conditional statements in their games to make the character bounce from one end of the screen to the other and then use conditionals to give the player the ability to control the character's position.

Lesson 6 - Work Day: The sixth lesson, if needed, can be dedicated to a work day to catch up on putting the character in the game and getting the character to move. If the students do not need a work day, the facilitator of the course can choose to skip this lesson and continue on with lesson 7. Our class was particularly fast partially due to the high ratio of teachers to students so we skipped the work day.

Lesson 7 - Falling Objects: The seventh lesson is used to describe the logic needed for falling objects. If the student's already designed and implemented the falling objects in their game during extra time, this lesson can be skipped. Since they already learned all of the specific logic needed (e.i. if statements, variables, and functions) the lesson is short as we just explain how they can apply their previous knowledge to this new problem. This allows the students to show how much they have retained from the previous lessons and work. *Lesson 7 Worksheet/Assignment:* The worksheet for lesson seven is simply a grid similar to lesson one. The students start by drawing what they want their falling object to look like on paper, then put it into a falling object function. They then add the logic to make it fall and reset to the top of the screen when it reaches the bottom.

Lesson 8 - Collision Detection: Lesson eight demonstrates how to add collision detection to the students' games. In this lesson we cover the definition and base logic behind collision detection and describe our solution of using bounding boxes. We then show the logic and math behind creating our bounding boxes around our characters. *Lesson 8 Worksheet/Assignment:* Instead of a worksheet for this lesson, we decided to begin working right away and providing teacher and TA assistance to students to put in the needed logic for bounding boxes in their games.

Lesson 9 - Win/Lose Screen: Lesson nine goes over the logic of creating a win or lose screen for the game. This lesson is very short because the students already knew all of the needed logic for creating this screen. We discuss the importance of booleans and conditional statements for the process. *Lesson 9 Worksheet/Assignment:* There is no worksheet for this lesson since they have already worked with all of the needed concepts before. The goal for their assignment is to both design a screen for winning or losing and then create a trigger to activate the screen when the player either wins or loses.

Lesson 10 - Demo Day: This class period is dedicated for the students to make any final adjustments to their games and add any fun features or designs to their game. They have the

entire class period to work on finishing their game and they are given the option to demo the game to the whole class, allowing other students to watch or play everyone's games. We helped the students work on any specific goal they want to accomplish for their game. This is a chance for students to be creative and add what they want. One student wanted to add levels so that the game goes faster after the player reaches specific intervals in their score and another student wanted to add a high score function to the game.

4 Results:

Though the class was a small sample size of three students, with one of them leaving during the third week of the class due to outside variables, the results of the course were overwhelmingly positive. Of the two students who finished the whole class, both completed fully functional games with additional features of their choice such as functionality to keep track of high score and adding levels. The students showed continued interest throughout the entirety of the course, even through the most difficult sections. We had two data points in the forms of pre-course and post-course surveys. As seen in figure 1, there was an increase in the students' belief that they have the ability to learn cs and there was an increase in one student's desire to consider a cs career; the surveys were given on a seven point scale. The student's finished games can be seen in figures 2 and 3.

	Pre-Course	Post-Course
	I Can Learn Programming	I Can Learn Programming
Student 1	2	4
Student 2	5	7
	I Would Like to Learn CS	I Want to Continue to Learn CS
Student 1	4	5
Student 2	5	6
	Consider a CS Career	Consider a CS career
Student 1	4	4
Student 2	4	5

Figure 1: Pre and Post Course Student Survey Data

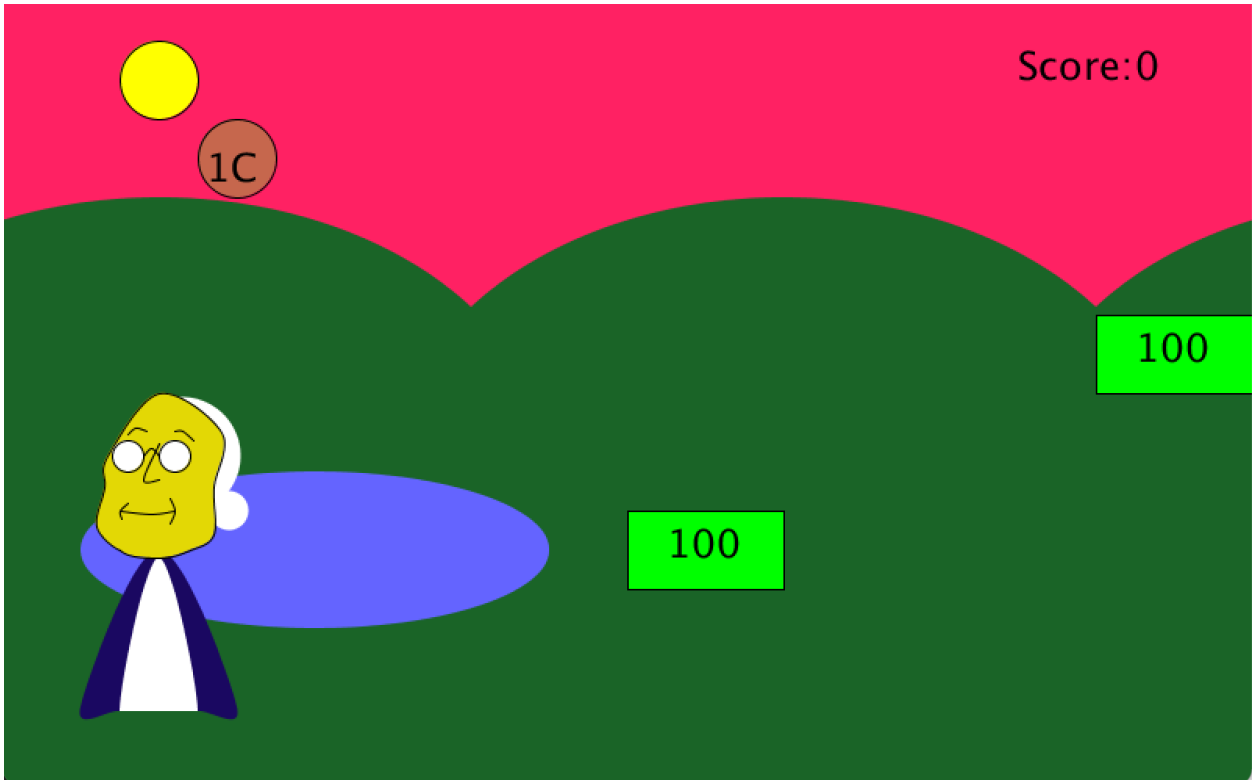

Figure 2: Screenshot of Student One's Game

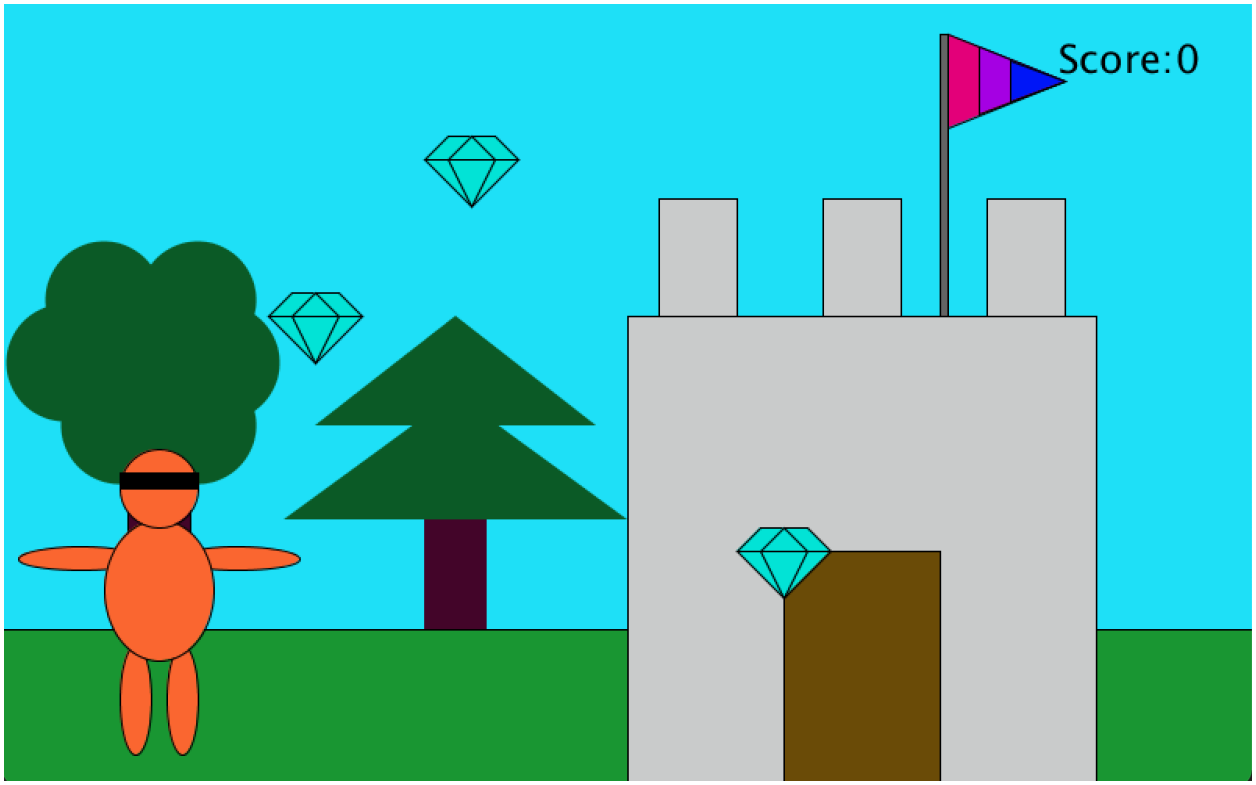

Figure 3: Screenshot of Student Two's Game

5 Reflection:

Our experience of teaching an introductory computer science course to incarcerated youth revealed various takeaways, both positive and negative. The strongest takeaway was the necessity of flexibility for the teachers. There are many uncontrollable variables when teaching a class in juvenile hall; students come and go, some days students have more or less interest in class, and the group of students would be different for any future iterations of this course. Our group of students were generally very engaged and motivated to create a personal and interesting game, however, other groups of students may be less engaged by game design. Due to the unknowables in both the larger group of students and also the smaller day to day variables, flexibility is a must. There were days when we did not use the powerpoints for lectures and only chose to do just one of the worksheet problems because students were less engaged on certain days due to outside circumstances.

Some additions to this iteration of the class that worked well were the inclusion of worksheets and base code. The worksheets were helpful for the students to work as a group to solidify their understanding of certain topics. We never had them work individually on the worksheets, but having the whole class work on one problem at a time together allowed us to see what they understood and what they did not so we could help them learn the needed concepts. The base code was also a huge success. The students had a lot less difficulties getting started and we removed a lot of the tedious and somewhat confusing work for the students so that they could instead put all their focus on the more engaging aspects of designing their game and programming the more functional aspects of the game. We did not plan for how efficient the base code would make the class, and ended up being a full week ahead of schedule by week three of the course, which is covered in the section below on some of the difficulties we encountered.

Though overall the class was a success, there were some aspects that could be improved on. One of the difficulties we had was with the schedule of the class. Because the students we were teaching were generally very engaged and understood the topics quickly and because the base code took out some of the longer tedious work, the students were all about a week ahead of schedule by week three. We decided to push all of our lessons forward by a week and create more goals for the students for the extra lesson times. Once we added in the extra goals the course continued smoothly, but it was just another example of how flexibility is a must have for teaching this course. Some of the worksheets were also too long, or too complex or both. For each class, we chose in the moment just one or two problems to solve as a class from the worksheet instead of doing all of the problems. The worksheets could be improved upon to be shorter and have every problem easier to understand for the students. In the format we did of working together through problems, the worksheets still served their function.

6 Future Work:

We hope for this course to continue on in the future. One of the main purposes of our intervention in this course was to create an easily repeatable class. We documented our lessons and lesson plans as well as the worksheets and all the base code needed. For anyone looking to create a similar course in different areas or continue on with the relationship between Cal Poly

computer science and the local juvenile hall, they can use the curriculum we created to reteach the same course. Since there are different students constantly moving through the juvenile hall, it is highly likely that there will never be the same students during any reteaching of the class, so the curriculum does not have to change in the future. As long as the teachers and TAs are flexible and prepared to go in to teach during the length of the course, it can be repeated easily.

7 References:

[1] Noble, Safiya Umoja. *Algorithms of Oppression: How Search Engines Reinforce Racism*. New York University Press, 2018.

[2] Camacho, Michelle M, and Susan M Lord. *Borderlands of Education: Latinas in Engineering*. Lexington Books, 2013.

[3] College Board (2019). AP Program Participation and Enrollment Data, 2019.

[4] Kapor Center and the Computer Science for California (CSforCA) coalition. [n.d.]. Computer Science In California's Schools: An Analysis of Access, Enrollment, and Equity.

[5] Tara J. Yosso * (2005) Whose culture has capital? A critical race theory discussion of community cultural wealth, Race Ethnicity and Education, 8:1, 69-91, DOI: 10.1080/1361332052000341006

[6] Sarita Yardi and Amy Bruckman. 2007. What is Computing?: Bridging the Gap Between Teenagers' Perceptions and Graduate Students' Experiences. In Proceedings of the Third International Workshop on Computing Research (ICER '07)> ACM, New York, NY, USA, 39-50. https://doi.org/10.1145/1288580. 1288586

[7] Kirsten Mork, Theresa Migler, and Zoë Wood. 2020. Introducing Computing to a Cohort of Incarcerated Youth. *In Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 234–240. DOI:https://doi.org/10.1145/3328778.3366820

[8] Theresa Anne Migler-VonDollen and Lizabeth T Schlemer. 2018. Engagement in Practice: Teaching Introductory Computer Science at County Jails. In 2018 ASEE Annual Conference & Exposition. ASEE Conferences, Salt Lake City, Utah

[9] Michael Panitz, Kelvin Sung, and Rebecca Rosenberg. 2010. Game programming in CS0: a scaffolded approach. *J. Comput. Sci. Coll.* 26, 1 (October 2010), 126–132.