PLTW COMPUTER SCIENCE

Problem 1.5.4

Design a *Python* GUI

Introduction

Many software applications are designed to allow people to be creative. Computing opens new opportunities to express ourselves artistically. What software would help you be creative?

Many software applications are for entertainment, allowing people to enjoy the creativity of the product's designers. What creative ideas do you have that could be implemented in software?

Materials

- Computer with Internet access
- Scratch or *Python*® Development Environments

Resources

1.5.4.PY stamper.zip

Reference Card for Tkinter

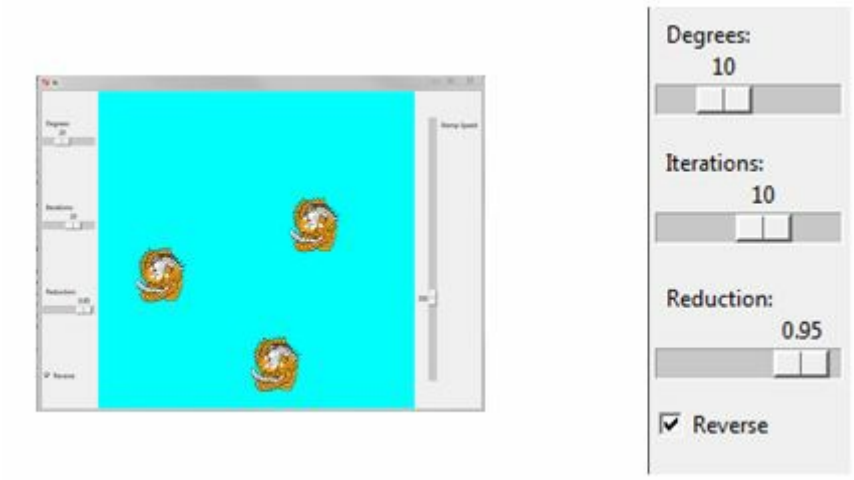
Problem 1.5.4 Rubric

Procedure

Part I: Being the Client

1. Greet your partner to practice professional skills. Review and record team norms for driver-navigator roles and version control.
2. Your teacher will demonstrate examples of GUIs used for design.
    - i.a Download and run `1.5.4 stamper.py`. Four versions of this program that show a progression of development are also available and may be useful when you are developing code.

- The Cornrow C



- The Cornrow Curves Software program, ©2003 Ron Eglash, is found at http://csdt.rpi.edu/african/CORNROW_CURVES/cornrow_software/cornrow_softwa

3. Review the following criteria for the client's request.

   Client #1

   - You work with a team of people in an organization. You want some software to help your organization's creative talent express themselves.

   Client #2

   - You work with a team of people that create games. Your product line includes strategy games and video games. Your products range from single player to social games. Your team is having a difficult time finding enough programmers so you are contracting out one of your best ideas for a game.

4. Initially you will create a software design problem as the client, and another team will act as your development team and work toward a solution. In choosing the assignment your development team will work on, you will follow a three-step process:

    - Brainstorm
    - Develop
    - Select

5. Brainstorm ideas using tag lines and thumbnail sketches. Follow the guidelines for brainstorming:

    - Never criticize ideas during brainstorming
    - "Piling on" is welcome
    - Aim for quantity, not quality or depth

6. Develop one or two of your ideas with further discussion and documentation.
7. Select one idea. Communicate your idea to a team of developers.

Part II: Being the Development Team

8. Half the class will now become development teams. As the client describes their idea to you, document what it is they want the product to accomplish.
9. Ask questions about what the client wants the software to achieve. Confirm that you understand what the client wants by describing it back to them while using the documentation you produced in the previous step.

    Waterfall Requirements Engineer: *“Two people can interpret human language very differently! You probably don’t yet understand what the client wants.”*

    Agile Developer: *“You probably don’t yet understand what the client wants. The client will be able to communicate more precisely once some of the project has proceeded.”*

    As one of the deliverables of this project, retain your documentation of the client’s requests.

10. Create a product backlog listing the work of the project in large chunks.

    Prioritize the backlog. You will work on the top item during the first sprint. You might want to break that top item into smaller pieces so that you will be able to finish at least one whole piece.

The top item in the first sprint often follows a **tracer route** through the layers of the application. For example, if your client wants a game that plays checkers, the developer's first item might be to connect the user input to the position of the checker pieces in the model. This would be created before beautifully designed checker piece graphics or sound effects.

Create

- A **product backlog** with the most important **user stories** at the top. User stories lower on the list can be large and poorly defined.
- The top one or two user stories from the backlog are broken into specific small pieces that form the **sprint task list**. Create the task list such that you think you stand a decent chance of accomplishing everything on the task list during the time allotted by your teacher.

  How much you get done per hour is your development velocity. Estimating how long a task will take is notoriously difficult for programmers early in their career. People get better at estimating how long a task takes if they practice estimating velocity.

  Your teacher will tell you how much total time you have for this initial sprint. Assign "small," "medium," or "large" to each of your sprint tasks.

As one of the deliverables of this project, retain a record of your plans from the beginning of the sprint.

11. Strategize, code, and test in small increments.
12. As one of the deliverables of this project, retain versions of your code at intermediate stages. Before moving on from these intermediate stages, go through your code comments and improve them. By spending time working on documentation as you develop code, you will avoid having a **technical debt**. Technical debt is when too much time is spent writing code and not enough attention paid to other crucial areas.
13. At the end of the sprint, reflect on how well your product meets the client's needs. Include descriptions on strengths and weaknesses of the solution as well as potential plans for the next sprint.
14. Prepare a high-level explanation of the interface for the user. The user is the intended audience of this explanation, so the code is not relevant.
15. Prepare to present your project at the end of the first sprint. Your teacher will describe the presentation format they would like you to use.
16. Reflect on the team dynamic. What helped the team work well together? Discuss your reflection with your partner and then write a reflection individually.
17. Deliverables:

    - Documentation of your understanding of the client's ideas

- Documentation of your planning: backlog and sprint task list
- Code - three or more versions, each documented
- High-level explanation of the interface, intended for the user
- Reflection on collaboration
- Practice Opportunity for the *Create* Performance Task

  "Create a 50-59 second video in which you demonstrate the running of at least one significant feature of your program." (Adapted from College Board Create Performance Task Part 1.)

- Practice Opportunity for the *Create* Performance Task

  "Identify the purpose of your program and explain what the video illustrates. (Approximately 150 words)" (Adapted from Create Performance Task Part 2a.)

- Practice Opportunity for the *Create* Performance Task

  "Describe the incremental and iterative development process you used, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated." (Adapted from Create Performance Task Part 2b.)

- Practice Opportunity for the *Create* Performance Task

  The main event loop is an algorithm to execute a variety of event handlers upon various user and program events. Your algorithm for the event loop combines the algorithms of each event handler that you created. “Describe how each algorithm within your algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. (Approximately 200 words)” (adapted from College Board *Create* Performance Task Part 2c.)

- Practice Opportunity for the *Create* Performance Task

  Explain how abstraction helped manage the complexity of your program. (Approximately 200 words)(Adapted from College Board *Create* Performance Task Part 2d.)

- Practice Opportunity for the *Create* Performance Task

  Paste your previous four responses into a new document and clearly label them 2a through 2d. Use screenshots to capture your entire program code, and paste the image(s) into the document, labeling the images 2e. Draw an oval around the segment of code corresponding to the algorithm of algorithms you described in deliverable (ix). Draw a rectangle around the segment of code corresponding to the abstraction you described in deliverable (x). Save the document as a PDF. (Adapted from College Board *Create* Performance Task Part 2e.)

Note: This last six deliverables are adapted from the official College Board Create Performance Task but they do not duplicate the content of College Board Task or Rubric. The task provided here contains elements that are different than the College Board Performance Task and Rubric. Please reference official College Board materials.

Conclusion

1. Assess your deliverables using the rubric attached and using the College Board *Create* Performance Task rubric.

2. Reflect on the creative process you used. What was useful? Discuss your reflection with your partner and then write a reflection individually.

Problem 1.5.4 GUI Design Rubric

	4	3	2	1
Statement of Client's Request	Artifact clearly explains what the client wants. A third party could interpret whether a product has met the requirements.	Many details are ambiguous or left out, but a third party could understand the client's request.	The artifact does not adequately communicate general terms of the client's request.	There is no evidence of the client's request.
Planning	Product backlog shows intent. Sprint task list subdivides to simpler subproblems.	Product backlog shows intent. Sprint task list shows subdivision to simpler subproblems, but some tasks are too big.	Product backlog and sprint task list show an inadequate attempt to indicate long-term intent and to break down problem into simpler tasks.	Product backlog or sprint task list are missing.
Executable Code	Code accomplishes objectives in a way that will make the code easy to reuse and maintain.	Code does not accomplish a sufficient task list. Intermediate versions demonstrate an iterative approach.	Code does not accomplish a sufficient task list. Intermediate versions demonstrate persistence, showing a variety of approaches that did not work.	Code does not accomplish a sufficient task list. There are not distinct intermediate versions demonstrating persistence or iterative development.
Other comments:				
Documentation of Code	Documentation of work is accurate and thorough. Contains: • Comments on sections of code • Comments in-line • High-level explanation of product intended	Documentation of work is accurate, but could be more extensive. Contains: • Comments on sections of code • Comments in-line • High-level explanation of product	Documentation of work is inaccurate or missing in many places.	Documentation is insufficient throughout.

	for the user	intended for the user		
Collaboration	Provides helpful original input to others. Promotes positive, productive, and respectful team dynamic. Encourages and incorporates input from others. Promotes equitable workload.	Provides adequate original input to others. Maintains positive, productive, and respectful team dynamic. Positively incorporates input from others. Maintains equitable workload.	Provides significant but limited input. Usually maintains positive, productive, and respectful team dynamic. Receives input from others. Shares workload somewhat equitably.	Provides limited input. Is not promoting positive, respectful, or productive team dynamic. Discourages or is unresponsive to input from others. Does not promote equitable workload.
Presentation	Demonstrates very effective presentation techniques: • Posture • Gestures • Voice • Eye Contact	Demonstrates mostly effective presentation techniques: • Posture • Gestures • Voice • Eye Contact	Demonstrates some adequate presentation techniques: • Posture • Gestures • Voice • Eye Contact	Demonstrates inadequate presentation techniques: • Posture • Gestures • Voice • Eye Contact