Standards Alignment

Game jams can be hugely varied in how they're run, what their outcomes are and what people can take away from them. As a result, we've compiled a list of some of the CSTA Computer Science Standards that could be applied to an example Game Jam. This won't be an exhaustive list but is a good starting point.

These standards may not directly apply to your school but should give you an idea of what areas of Computer Science teaching can be included in a jam.

In this example, our theoretical jam will see Middle School students creating a game that uses their own assets. Students are also required to document their game through a Design Document and thorough commenting on the game's code.

CSTA Standards

2-AP-10 - Use flowcharts and/or pseudocode to address complex problems as algorithms.

Complex problems are problems that would be difficult for students to solve computationally. Students should use pseudocode and/or flowcharts to organize and sequence an algorithm that addresses a complex problem, even though they may not actually program the solutions. For example, students might express an algorithm that produces a recommendation for purchasing sneakers based on inputs such as size, colors, brand, comfort, and cost. Testing the algorithm with a wide range of inputs and users allows students to refine their recommendation algorithm and to identify other inputs they may have initially excluded.

2-AP-11 - Create clearly named variables that represent different data types and perform operations on their values.

A variable is like a container with a name, in which the contents may change, but the name (identifier) does not. When planning and developing programs, students should decide when and how to declare and name new variables. Students should use naming conventions to improve program readability. Examples of operations include adding points to the score, combining user input with words to make a sentence, changing the size of a picture, or adding a name to a list of people.

2-AP-12 - Design and iteratively develop programs that combine control structures, including nested loops and compound conditionals.

Control structures can be combined in many ways. Nested loops are loops placed within loops. Compound conditionals combine two or more conditions in a logical relationship (e.g., using AND, OR, and NOT), and nesting conditionals within one another allows the result of one conditional to lead to another. For example, when programming an interactive story, students could use a compound conditional within a loop to unlock a door only if a character has a key AND is touching the door.

2-AP-17 - Systematically test and refine programs using a range of test cases.

Use cases and test cases are created and analyzed to better meet the needs of users and to evaluate whether programs function as intended. At this level, testing should become a deliberate process that is more iterative, systematic, and proactive than at lower levels. Students should begin to test programs by considering potential errors, such as what will happen if a user enters invalid input (e.g., negative numbers and 0 instead of positive numbers).

2-AP-18 - Distribute tasks and maintain a project timeline when collaboratively developing computational artifacts.

Collaboration is a common and crucial practice in programming development. Often, many individuals and groups work on the interdependent parts of a project together. Students should assume pre-defined roles within their teams and manage the project workflow using structured timelines. With teacher guidance, they will begin to create collective goals, expectations, and equitable workloads. For example, students may divide the design stage of a game into planning the storyboard, flowchart, and different parts of the game mechanics. They can then distribute tasks and roles among members of the team and assign deadlines.

2-AP-19 - Document programs in order to make them easier to follow, test, and debug.

Documentation allows creators and others to more easily use and understand a program. Students should provide documentation for end users that explains their artifacts and how they function. For example, students could provide a project overview and clear user instructions. They should also incorporate comments in their product and communicate their process using design documents, flowcharts, and presentations.