Guidelines for PECC-Activities

Items	Tasks	PECC Category	
	STAGE 0: Preparation		
(0.1) Structure of the coding course	**0.1.1: Choose your target group** • Gender, age (12-15 years old) • In or outside the classroom (after school program) • Youth center, coding camps, etc.	**Coding – Structure**	□
	0.1.2: Determine the available units Recommendation: • Instruction (the Starter, 1-2 units) • Game design (The Main Learning, 2-4 units) • Coding (The Main Learning, 4-8 units) • Presentation (The Closing, 1-2 units)	**Creativity – Structure**	□
	0.1.3: Choose suitable tools **For coding:** • Visual-based coding (e.g., Pocket Code, Scratch, Snap) or robotics (e.g., Lego Mindstorms) • Text-based coding (e.g., Text Editors, Eclipse, Android Studio, etc.) **For creating game assets (how they are produced)*** • Artwork (by hand) • Tools, e.g. Photoshop, InDesign • Assets from the internet (be aware of copyright issues!) • Personal photographs • Use assets from available media libraries • Sound design: personal records, internet		□
	0.1.4: Define the learning goal(s)🔔 or a general goal for games A (learning) goal consists of three parts: action, content, and condition. **The (learning) goals need to be defined according to 🔔** **(a)** Learning goals/objective of the curriculum subject in which coding is applied Example: "Add 5 questions about the `French Revolution' to your game". **(b)** Learning goals for game design/coding Example: "Integrate min. 2 objects designed by yourself (artwork)"	**Coding – Teaching Approach**	□
	0.1.5 Choose the engagement level * Group constellations (homogeneous/heterogeneous teams) (1) small groups (2-5) (2) pair work (3) work individually (but all working on the same learning goal)		□
(0.2) Prepare your material	**1.1.4 Create tailored challenges *** (a) Template/Framework: students start with a pre-coded game and to add code/assets to finalize it (also allows customization, etc.) (b) Learning-by-doing: provide tutorials, helpful material/prepared functions, guidance ✏	**Engagement – Collaboration**	□
	1.1.5 Set-up & Prepare • Presentation, if needed • Print storyboards (→ see storyboard)	**Coding – Structure**	□

	• Handicraft items for “brick on paper” activity (scissors, tape, paper, etc.) • Template/framework program, example games to present, etc. • Setup platforms/tools (accounts, installation, charge mobile devices) • Other (room, date, time, equipment e.g., projector, etc.)		
STAGE 1: Introduction			
(1.1) Create a realistic picture of STEM jobs	**1.1.1 Create a safe environment *** Allow/ask questions, spark discussions In small groups/with the whole class: • Which technical professions do you know? • Which study/training do you need to acquire technical skills? • What does a computer scientist do? Do you know people who are working in those fields? • Who already has experience in coding? Which tools did you use for coding? • What is coding? What is an algorithm? • Which programming languages do you know? **1.1.2 Visit companies, invite role models *** ✂ **Be a role model/mentor on your own!** • Asking for resources to promote the improvement of technological knowledge (companies, universities) • Establishing direct communication between STEM professionals and students • Invite STEM professionals (role models) from the industry/university • Tell about role models and famous women who have succeed in computer science (e.g., Ada Lovelance) • Inspire students for STEM • Address the issue: Why do you think there are fewer women in IT than men? **1.1.3 Understand the learner’s playing behavior *** • What kind of games do you play? • What makes you play games? • What hinders you from playing games?	**Engagement –** **Warm up** **Playing - Play**	□ □ □
(1.2) Provide a convenient starting point	**1.2.1 Design Learning: What is/How do _________?** Students are not familiar with “coding vocabulary” and practices. The most important terms are (→ IT Glossary) • Loops, conditions, variables, data types, objects, pseudocode, conditionals, function, iteration, parameter, broadcast messages, etc. It is not necessary to explain all of them, ask them if they are familiar with these concepts. Explain why they are needed (e.g., for creating a score, you need to define a variable; in order for objects to interact, you need messages). **The answer is: Engagement!** (a) integrate important functionalities in the example program (next step), so students can see what they are needed for (b) prepare a presentation with showcases/example programs (c) do “Unplugged Coding”, e.g.	**Engagement –** **Collaboration** **Coding –** **Structure**	□

	○ "Program" a classmate like a robot (start/end point) ○ Paint "instructions" ○ Pack a rucksack with "variables" ○ Send "broadcasts" through the classroom		
	1.2.1 Introduce the tools or let students explore * • Show the UI, menu, and structure of the tool/platform • Show them where to find help, tutorials, useful forums/groups, demos (e.g., on YouTube) Coding: Starter program: ✂ (1) Create a collaborative program with the whole class (e.g. on the projector) which covers important steps (about the program they are going to create on their own): e.g., add an object, movement, interaction, etc. (a) One or two students come to the front of the class and add one small but meaningful step to the game (class is allowed to help) (b) Ask students for the next step while programming (2) Let students program a game (a small starter task) with the help of tutorials (guides, step-to-step) and let them add enhancements, e.g., add an animation, add a sound, score, etc.	**Coding – Personal Experiences**	□
	1.2.2 Don't forget the fun! - Let students PLAY * ✂ (a) Show students example games (b) Let them play games on their own (i.e. featured games, best practice)	**Playing - Play**	□
	STAGE 2: Story & Game Design		
(2.1) Foster self-directed learning to create personal experiences	**2.1.1 Bring "Freedom of Choice" to your course to create a sense of ownership *** • Designing of personal games from scratch: (a) Don't restrict the game design at all, let them choose game elements, e.g. story, genre, theme, goal, MDAs, assets (b) Define a frame, e.g., use of certain properties, genre, design elements, or MDAs • Use of templates: allow customization, personalization, and enhancements	**Playing** **Engagement** **Creativity** **Coding**	□
	2.1.2 Let's get it started! * • Describe the activity: task, structure, units –> strive for mutual understanding • Explain the (learning) goal: define a sub-goal for each unit • Support the formation of homogeneous groups	**Coding – Coding**	□
(2.2) Bring in the gaming/design elements	**2.2.1 Give students a storyboard** ✂✎ A storyboard (→ storyboard) could help students in their game design process, the template refers to the "Shape of a game" • Ask students to give their game a name • Let them tell a story	**Playing – Game Design**	□
	2.2.2 Classify the game: genre/theme/goal * ✎ • **Choose a genre**: ○ Action (platform/jump'n'run, shooter) ○ Adventure (RPG, text adventure/storytelling) ○ Puzzle (skill game) ○ Quiz ○ Simulation (racing, real-life)		□

	o Strategy • **Choose a theme**: o Criminal/detective stories, o Science fiction, fantasy, comic o Romance o Nature, animals, sports o Future, space o Realistic o Horror, etc. • **Choose a goal**: o Capture/destroy/avoid e.g., items or opponents o Territorial/knowledge acquisition, collection, e.g., items o Solve a puzzle or a crime o Chase/racing/escape something or somebody o Spatial alignment: positioning of elements o Build a character, resources o Negation of another goal: games end if the play act against the rules o No goal (e.g., storytelling, retelling, animations)		
	2.2.3 Who is the "star" in the game? * ✎ • Main characters, e.g., animals, fantasy figures, man/woman, boy/girl, items, transport, food, etc. • Side characters • Name all the characters to promote ownership • Background (i.e., theme) • Interactions between characters and their level of control		□
	2.2.4 Bring the games to LIFE (use MDA) * **Mechanics → Dynamics** • Points/rewards: e.g., earning points/currency to levelling up (reward completion of activities) or for a high-score list • Status/levels: thresholds or milestones that a player must achieve in the progression. • Challenges/achievements: tasks or actions users have to perform to be awarded • Virtual goods/self-expression: non-physical, intangible objects the user can, for example, exchange in virtual shops to customize their avatar • Leaderboards/competition: scores and rankings of users relative to others (e.g., high-score list) • Notifications: provide feedback for the user • Timer: set a time limit for actions **Aesthetics: provide visual, audio, and fantasy elements** • Sensation: create something completely unfamiliar • Fantasy: build imaginary worlds • Narrative: tell a story • Challenge: to master something • Fellowship: the player is part of a community • Discovery: the players need to explore • Expression: use individual creativity		□
	2.2.5 Get the games in shape! "Ceremony" ✎ • Title screen: name of the game • Introduction screen: explain the goals and rules (mechanics) of the game • Game screen(s): 1-n levels • End screen: game over or win screen		□

(2.3) Let students be creative and express themselves	**2.3.1 Foster students' sense of ownership - It's their game! *** • Also within templates/frameworks! • Edit/change, customize and personalize: assets, characters, looks, backgrounds and screens (shape of a game) • Add sounds, record media • Suggestions: ○ Use art lesson for design session ○ Have group members already started to code? No problem: Let them *change* roles after a while!	**Creativity – Freedom of Choice**	□
	Stage 3 – Coding		
(3.1) Now let's start coding!	**3.1.1 Tinkering activities: First … on paper! ✄** • Pseudocode: students should think of commands, variables, etc. they will need for their games • Hands-on/bricks on paper: print out the bricks, students add them to their objects ○ Where to place the objects? ○ Which size they are? ○ How I will control my objects? ○ How and who should interact/communicate with each other? ○ How will I use MDA in my game? ○ Where to define my variables?	**Coding – Personal Experiences**	□
	3.1.2 Ready…Set…Code! * • Students should try it out and see what happen, e.g., If something does not work like expected: change it • Consider failure as part of the learning process • Do not show/explain all at once: break down the content into sub-goals for every units	**Engagement – Collaboration**	□
	3.1.3 Repeat, focus, and foster collaboration * At the beginning of every unit: • Let students repeat what happened in the last unit and present sub-goals for today's unit. • Ask: What was difficult? What was easy? Open questions? • Observe the teamwork: enable students to assume different identities and roles (leader, designer, programmer, etc.). • Build confidence: ○ Praise students, provide confirmation ○ Celebrate "Aha!-effects" ○ Provide recognition of work done ○ Balance extrinsic and intrinsic motivators • Support collaboration and communication during the whole game production process • Foster originality and self-expression ○ What else can you add to the game? ○ Are you satisfied with your game? ○ Is anything missing? ○ Is there some room for improvement? **→ Students feel pride/self-efficiency!** • Check the state of the work: ○ Who needs more time? ○ Provide extra tasks for the faster ones.	**Creativity – Freedom of Choice**	□

(3.2) Don't forget the gender	**3.2.1 Be gender-sensitive/aware *** • Be sure your learning materials are free of gender stereotypes (example games, learning goals, templates/frameworks). • Use a gender sensitive language, e.g., do not foster male masculinity in tech (e.g., only refer to a technician as HE), consider that language forms pictures; so make women visible and audible, use both definitions (e.g., in German) or more neutral forms if they exist. • Use gender sensible language and imaginary for slides, material, and examples. • Praise students the right way: not for spent effort/time, but for their knowledge. • Provide a stress-free and anxiety-free working environment by considering different skill levels or preferences. • Ensure a competition-free environment. • Observe groups/individuals: who is engaged, who asks, and who is holding back. • Support (girls) to pursue and persist in technology	**Engagement – Collaboration**	□
Stage 4: The Closing			
(4.1) Enable recognition of the student's progress by peers, teachers, and parents	**4.1.1: Allow students to present their games in public to provide a sense of ownership & pride** (voluntary/mandatory) • In front of their peers (during the last unit) • At events (e.g., open house days, final event) • Sharing (i.e. through a public forum) • Recap session / ask questions: ○ Who will program at home? Tell his/her friends? ○ Highlights/problems, etc. **4.1.2: Make a short quiz** ✄ • Discuss the questions at the beginning of the unit with the whole class • Define easy questions, e.g., single choice questions • No teamwork, no grading • Discuss the questions after the quiz **4.1.3: Evaluate submitted programs** 🔔✄ Assessment of: • Confirmation of achievement of the learning goal(s) • Use of game design elements • Program structure (e.g., code statistics, finished program) (→ assessment template sheet)	**Engagement – Collaboration** **Coding – Structure**	□ □ □

Hints: per unit at 45 minutes (1) (2) (3): steps (a) (b) (c): choose
Legend: * gender sensitive 🔔 only for schools ✄ optional ✎ only if coding from scratch