C29 SD:ST Assignment3 – Doomenstein Prototype

Prof. Eiserloh

# **Requirements / Grading Rubric**

* (8) Data/Definitions/EntityTypes.xml is parsed, creating registered-by-name EntityDef instances( 1h
* (8) <Entities> section of map .xml files are parsed; each entity referenced is instantiated from named definition and spawned in the map( 1h
* (3) Three new Entity subclasses: Actor, Projectile, Portal. Projectile and Portal must exist and be instantiable, but are not used in A3.
* (5) Map abstract base class offers two factory functions for spawning new entities:

class Map

{

…

virtual Entity\* **SpawnNewEntityOfType**( std::string const& entityDefName );

virtual Entity\* **SpawnNewEntityOfType**( EntityDef const& entityDef );

* (5) Map abstract base class owns all Entities (which can be of type Entity, Actor, Projectile, or Portal) in that map, in lists that it manages.
* (5) Game uses a pointer to a particular entity which is the entity “currently possessed by the Player” (or nullptr in “ghost” camera mode).( 1h )
* (10) F3 toggles in and out of “ghost” mode;( 1.5h )
  + if Game::m\_player == nullptr, we are in “ghost” mode; no entity is possessed; keyboard/mouse controls camera directly.
  + else, Game::m\_player points to the entity we currently “possess”; keyboard/mouse controls that entity instead, and the camera is snapped (at the end of update, after all movement and physics) to the entity’s current eye position and orientation.
  + If F3 is pressed while in ghost mode, the “nearest entity in a forward 90o sector of the camera and within 2.0 distance3D” is possessed.
* (5) Each Entity has four physics properties:
  + bool m\_canBePushedByWalls = false;
  + bool m\_canBePushedByEntities = false;
  + bool m\_canPushEntities = false;
  + float m\_mass = 1.f;
* (5) Entities have entity-vs-entity physics (done first) as well as entity-vs-wall physics (done second), reflecting each Entity’s physics properties.
* (10) A game-custom 2.5D Map::Raycast method which can report information about solid tile and/or entity first impacted.
* (8) Multiple different game-specific billboarding methods are supported in game code.
* (10) Entities are drawn using billboarded sprites referenced in the EntityTypes.xml data file; each data entry can control: (2h
  + The method of billboarding (CameraFacingXY, CameraOpposingXY, CameraFacingXYZ, CameraOpposingXYZ) to use;
  + The sprite coordinates to use for that entity;
  + Additional sprites to use, if any, for alternate view angles (e.g. from the side, rear).
* (8) Each entity billboard chooses the sprite whose stated view direction best matches the local-space displacement from entity to camera.
* (10) If debug drawing (F1) is enabled, each entity also debug-draws itself as a cyan wireframe cylindrical cage representing its physical body. If the player is currently possessing an entity, also debug draw the forward raycast impact position and impact normal in world space.

# **Submission**

*See “Submitting Assignments” in the course syllabus (available in Canvas) for details on assignment submission.*