CS 351
Design of Large Programs
Zombie House

Instructor: **Joel Castellanos**

e-mail: joel@unm.edu

Web: http://cs.unm.edu/~joel/

Office: Electrical and Computer Engineering building (ECE). Room 233

2/23/2017

Overview

- Group project: Three students per team.
- Tuesday March 21 in class (week after spring break).
- Third person, single player cooperative game with JavaFX for 3D rendering with texture, lighting and material effects.
- Player is:
  - Player is in a house full of zombies.
  - Must find exit before his or her brain is eaten.
- Game contains procedurally generated house.
- If the player's brain is eaten, the level is restarted back in time, but splits with his or her past self. All past instances of the player repeat the past with the new instance controllable by the player.

2

Past and Present Self (1 of 2)

1) The player will need to cooperate with his or her past self.
2) Your past selves have no AI. Each does exactly what it did before up to the moment of its death. At that point, it falls to the floor.
    a) Sometimes this might mean that your past self is swinging at a zombie and/or takes damage from a zombie that is not present.
    b) Sometimes this might mean that your past self walks right through a present zombie.

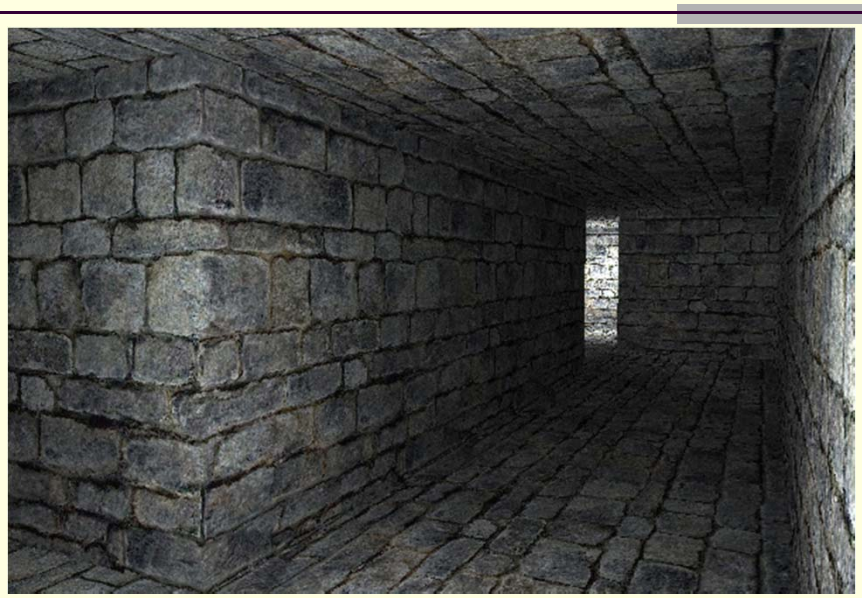
3

Past and Present Self (2 of 2)

- Zombies attack your past and present selves following the same AI rules with no distinction between the two.
- Your past selves do not take damage.
- Your past selves attack as they did in life. If and only if a zombie or your present self is in range, it takes damage.

4

Sample view of Exit with Player Light



5

Procedural Generation

- In computing, procedural generation is the method of creating data algorithmically rather than manually.
- In computer graphics it is commonly used for creating textures.
- In video games it is used for creating various kinds of content such as items, quests or level geometry.
- Procedural generation is a stochastic process that uses random number generators, yet is not simply random.

6

Procedurally Generated House

Each new level and each time the program runs, the game must procedurally generate a “House” that:

1) Has a rectangular perimeter with a 2D floorplan.
2) Is connected (every location is reachable).
2) Has exactly 1 exit in a procedurally generated location along an outer wall.
3) Has at least n rectangular rooms, m hallways and p obstacles (square pillars, mid-room walls, …) in procedurally generated locations, with procedurally generated connections and procedurally generated sizes.
4) Is space filling (no interior unreachable places).
5) Has no doors.

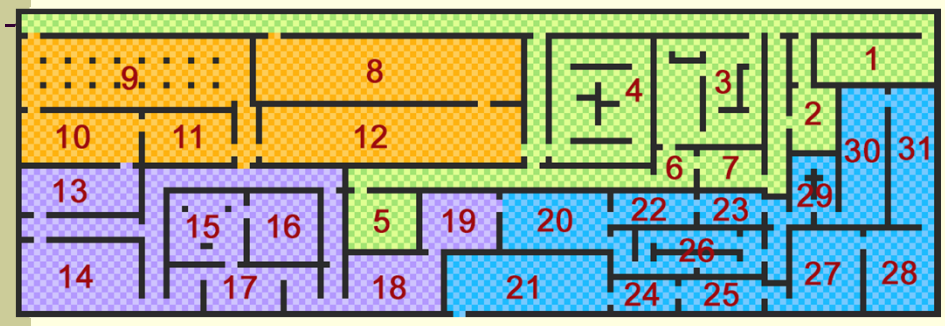
7

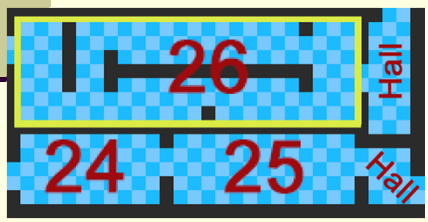
The House

- The house is one level.
- Each level must use a number of visually different floor and wall textures.
- All floor tiles of the same texture must be ***connected***.
- ***All paths*** from the player spawn spot to the exit must pass through at least one tile of at least three distinct texture region.
- Every gap between walls/objects must be large enough for any moving elements to pass through.
- Player always spawns in a hallway.
- Zombies never span in hallways.

8

Example Floor Plan for 31 Room House





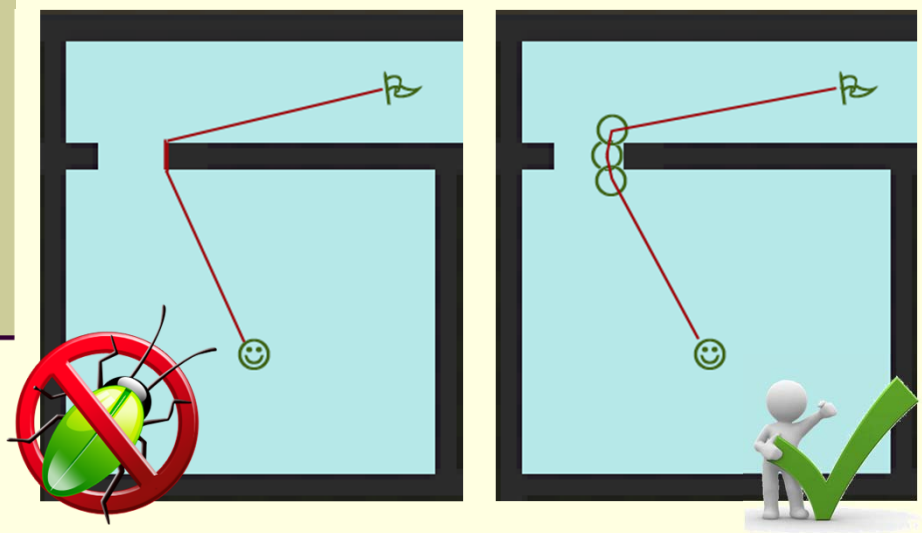
House (& rooms) must be ***rectangular***.

Obstacles cannot completely divide a room.

Which hallways are legal player spawns?

9

Shortest Path: Zombies Have Size > 0



10

Game Attribute Settings

- Where attributes are specified, they must be followed. Where variables are given (such as the house must have n rooms), it is left to each development team to choose values that 1) make it possible to meet other requirements and 2) make a fun and balanced game.
- "Fun" and "balanced" are, within limits subjective. Some of you will like harder / more complex settings then others. However, there is a wide acceptable range, it is far from infinite.

11

Update and deltaSeconds

- The “main game loop” must execute at no less than 30 Frames per Second (fps) on a 3rd floor ECE computer.
- Ideally (for full credit), the main game loop should execute at 60 fps. Do not run faster than 60 fps, even if you can.
- These are average times. The actual time between updates will fluctuate with processor load.
- deltaSeconds is the actual change in wall-clock seconds between the current and the pervious frames.

12

Attribute: Player Hearing, Default = 20

1) Whenever a zombie walks, if the player is within a Euclidean distance of **`PlayerHearing`** tiles, then sound effects must be played (some mix of footsteps, sliding, bumping, dragging and/or groaning). Calculate the Euclidean distance ignoring objects and walls .
2) The ***volume*** of the ***loudest*** channel of the sound effect must decrease with distance.
3) The difference in volume between the left and right channel, both for zombie footsteps and wall hits, must follow an algorithm chosen by the group to the effect of indicating the direction from the player to the reporting zombie.
4) If multiple zombies are making noise within hearing, merge the individual contributions before playing the sound.

13

Attribute: Player Speed, Default = 2.00

1) The player's movement must be controlled by ASWD.
2) The camera direction must be controlled by the mouse.
3) The player's walking speed is **`playerSpeed`** tiles per sec.
4) The player moves only while one or more movement keys are depressed.
5) If **`playerSpeed`** = 1.0, and the player depresses the W key for 1.23 seconds, then, assuming there is not an obstacle in the player's way, the player moves forward 1.23 tiles.
6) If more than one movement key is pressed, the player's movement direction is the vector sum of the component movements, but the magnitude must still be **`playerSpeed`**.
7) The player stops when colliding with a wall or object.

14

Attribute: Player Stamina, Default = 5.00
Player Regen, Default = 0.20

1) While moving, if the player presses the '**`shift`**' key, then the player moves **`playerSpeed`** × 2.0 tiles per second.
2) While running, the **`playerStamina`** is decreased by the elapsed time in seconds.
3) The player stops running when:
   a) The '**`shift`**' key is released, or
   b) No movement keys are pressed, or
   c) **`playerStamina`** reaches 0.
4) When not running, **`playerRegen`** × **`deltaTime`** is added to **`playerStamina`** up to a maximum of the original **`playerStamina`** attribute for the level.

15

Attribute: Zombie Spawn, Default = X

1) While generating a level, each ***empty room floor tile*** has a change of spawning a zombie. On a given empty room floor tile, a zombie is spawned if a uniformly distributed random number on the interval [0.0, 1.0) is less than **`zombieSpawn`**.
2) Zombies do not spawn on hallway tiles.
3) If a zombie spawns on a location there is a 50% chance that it will be a Random Walk zombie. Otherwise it is a Line Walk zombie.

16

Attribute: Zombie Speed, Default = 0.5

1) Each zombie moves every update ***along its current heading*** a distance equal to:
`zombieSpeed` × `deltaSeconds` tiles.

2) Each zombie starts the game not moving and with a random heading.

17

Attribute: Zombie Decision Rate, Default = 2.0

1) While a zombie moves every update, it may only change its heading once every `zombieDecisionRate` seconds.

2) This is not effected by the zombie smelling a player nor by the player moving so that what was the shortest path on a past turn is no longer the shortest path. Zombie brains only evaluate shortest path on a decision frame.

18

Attribute: Zombie Smell, Default = Y

1) If a zombie's distance from the player is ≤ **`zombieSmell`**, then the zombie can smell the player. This distance is the the shortest-path distance (NOT the Euclidean distance ignoring objects and walls as is the player hearing).
2) If a zombie can smell a player, then the zombie "knows" the player's exact location and the shortest path, avoiding all obstacles, to the player.
3) If either a Random Walk or Line Walk zombies smells a player, then on the next decision update, the zombie will calculate the shortest path and adjust its heading to match.

19

Attribute: Health and Damage

1) The zombies attack with hands and/or clubs.
2) The player attacks with some hand weapon of your choice (knife, club, mace, sword, staff, nunchucks, gun...).
3) Each type of zombie and the player has a start health and a damage / second. (where the actual damage per update us based on the delta wall-clock time).
4) Each developer team must discover values for these numbers which work well with the various other parameters the team has chosen.
5) During each timeline in the game, damage done is permanent. Each time the timeline resets, all health of all game elements is restored.
6) When a player or zombie reaches zero health, it falls over, explodes, or whatever, but it must stop moving and damage.

20

Random Walk Zombie Intelligence

1) If a Random Walk zombie does not smell the player, then, on each decision step, the zombie will choose, with uniformly distributed probability, a heading 0.0 through 360.0 degrees from east.
2) After choosing a heading, the zombie will continue to move in that heading until the next decision update.
3) When a zombie collides with a wall or obstacle, the collision detection must stop the zombie at the wall. However, the zombie will continue to attempt to walk in that same direction until its next decision update.
4) On a decision update, if a Random Walk zombie had on the previous update hit a wall, then the zombie will not choose to move in that same direction.
5) On a decision update, it a Random Walk zombie is adjacent to a wall, but did not hit that wall on the previous update, then the zombie will not favor nor disfavor that direction.

21

Line Walk Zombie Intelligence

1) On a decision update, if a Line Walk zombie smells the player, then it will set its heading to the shortest path to the player.
2) On a decision update, if a Line Walk zombie hit a wall or obstacle on the previous update, then it will choose a uniformly distributed random heading different from its current heading.

22

Master Zombie

- Every level must include exactly one Master Zombie placed during the procedural level generation.
- Whatever any zombie detects the player, the master zombie also detects the player.
- The master zombie has one or more other special skills as defined by each group that perhaps varies with different levels.

23

Miscellaneous Requirements (1 of 3)

1) Graphics must be smooth and without flickering.
2) When the player walks, a soft walking sound must be heard.
3) When the player runs, a slightly louder running sound must be heard.
4) The game must use different sound effect for zombie and player walks.
5) The zombies are blind and deaf.
6) Neither zombies nor the player can pass through or occupy the same place as one another, walls or objects.
7) You may add additional key bindings.

24

Miscellaneous Requirements (2 of 3)

8) The game is third person and therefore, the player is a 3D model with articulated joints.
9) The zombies must be a 3D model with articulated joints.
10) There must be a noticeable difference in appearance between the three zombie types (could be just color).
11) All sound and graphic elements may be created by you or may be produced by a third party offering them free for non-commercial use. Of course, list the source of any third party material.

25

Miscellaneous Requirements (3 of 3)

12) When presenting in the classroom, the room sound system will be used. When I am grading sound effect details, I will use Bose SoundLink Around-Ear Headphones so I will be able to hear subtle directional effects you take the time to code.

26

JavaDoc for non-private classes

1) Program Level: Overall design and of each subsystem.
2) Class Level:
   a) What does this class do?
   b) How is it used by other classes?
   c) How does it use other classes?
3) Method Level:
   a) What does it do?
   b) What inputs (both arguments and global)?
   c) What assumptions does the method make (i.e. what are the expected ranges of inputs)?
   d) What return values and/or side effects?
   e) If nontrivial, what algorithm does the method use?

27

Quiz: The Strategy Pattern

What is the Strategy Pattern?

The ***Strategy Pattern*** defines a family of algorithms, encapsulates each one and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

28

Derived Requirements

Derived Requirement are requirements that are implied or transformed from higher-level requirement. For example, an ***explicate stakeholder requirement*** for long range or high speed may result in a ***derived design requirement*** for low weight.

Should the development team be sure to clarify all derived requirements with the stakeholders so as to document them as part of the formal ***requirements document***?

When, if ever, is it safe to assume?

29

Zombie House: Derived Requirements

- At least 3 diff't regions
- Some attack input mechanism or intuitive attack condition
- Limited resource-intensive functions (to meet 60fps)
- Zombies have reasonable HP
- Reasonable game length
- Display of player/zombie stats

30

Coding Style

- All code authored by members of a single group must follow a consistent set of formatting standards.
- As much as reasonable, code must be "*self-documenting*".
- Only use non-private methods and fields when there is good reason.
- Create well named methods for all logical units.
- Minimize code duplication.
- Have well defined and minimal ways in which objects interact.

31

Grading Rubric: 450 points (150 / member)

You are given existing code that, already meats many of your requirements.

You may or may not choose to use any or all of that code, but the grading rubric will be weighted toward the new requirements:

Adding polish to path finding, Zombie AI, player movement, procedural generation, ...)

Adding the single-player co-operative elements.

32

Grading Rubric: 450 points (150 / member)

150 Points: Past lives work within bounds of specifications.

150 Points: Graphics and Sound are smooth and significantly improved / extended from given code.

150 Points: Game mechanics are polished and significantly improved / extended from given code.

33

Additional Grading

- Self/Peer Evaluation Form (10 participation points for completeness and accuracy). This also effects the member distribution of project points.
- Oral Presentation (40 participation points person).
- Periodic Code-walk through and design presentation. Some during lab, some during lecture (10 participation points each).

34

Oral Presentation

- Tell us who you are.
- One person "drive" while other(s) talk. Optional: Swap roles.
- Demo the running game.
- Tell us what works and what does not work.
- Explain how you ***verified*** you meet the requirements of the:
  1) Floor plan.
  2) Zombies (random walk, line walk, shortest path, smell distance and collision).
  3) Sound effects.
- Tell of anything you are particularly proud.

35