Ian Keyworth (keywoi@rpi.edu)
Ray Coulter (coultr@rpi.edu)
Rory Perner (perner@rpi.edu)
Chris Mui (muic@rpi.edu)

Game Design Document for *Nova Radix*

I. Artist Statement/Philosophy/The WHY Factor (why create this game? why would someone want to play it?)

There are several reasons people would play *Nova Radix*. For one, people like simple games they can pick up as a distraction from homework, and this is easily one of those games – so long as the opening levels teach the essential skills well enough. People also generally enjoy physics. While this is not the most physics intensive or accurate game, mastering and manipulating momentum and gravity are fun. The mechanic is also interesting, and while not entirely new, this execution opens many design paths and offers a challenging and fun break from reality.

II. Predecessor or previous games/ distinctive factors in this genre

The game was most inspired by *Shift*, a flash game developed by Armor Games. In *Shift* game, you were given the power to inverse the floor and background (so one became the other) and the level turned on its head. Mechanics and level designs are partially inspired by *Portal*, by Valve, which has pushed the realm puzzle-games into a new dimension.

During our development process, *Shift 2* was released. One of the new mechanics introduced was to rotate the room. Fortunately for us this was only done at specific squares and in specific directions, so despite the fact that the square room rotated and gravity changed – like in our game – our concept remained mostly safe.

There have been rotation games done in the past, such as "Spin the Black Circle," developed by BubbleBox.com, where you have free rotational control of the maze – to any degree, mind you, so more precise than the *Nova Radix* – to move a ball to the exit through gravity while avoiding spikes and other obstacles. *Nova Radix* is different from this mainly in the sense that the player has total control over themselves, whereas in Black Circle it can get incredibly frustrating when you don't spin the room just right for the ball to avoid destruction, despite having a more interesting physics engine.

And Yet It Moves, an independent game sponsored by ImpulsProgramm, also rotates the world map 90 degrees to solve puzzles. The difference here is that the game plays much like the typical scrolling platformer, where the player only sees a small portion of the map at any given point; you are also able to die by falling too far or for too long. Nova Radix is a more direct puzzle game in the sense that the entire world is viewable at once, so the player can plan the whole route to victory without pushing a button. The puzzle oriented goal of *Nova Radix* also means that dying from falling would be more frustrating

since it is not as foreseeable, and the focus should be on avoiding the clearly marked hazards instead of the potential perceptions. The main goal is to finish the puzzle, not to "not die".

III. Target Audience

Anyone who likes games that are easy to pick up, or mentally challenging and creative. Although this could theoretically apply to any age beyond the point at which children are capable of understanding spatial orientation and gravity, the game is more focused towards a teenage sort of audience. The most evident result of this is the character, since the concept art portrayed a more intense yet loose appearance, reflective of teenagers. The gameplay itself, however, has a much broader range since it twists the physical concept of gravity that we all fundamentally understand, and in that sense we hope as many people as possible can enjoy the game.

IV. Introduction & Story

Though there is no defined story, the game takes place on some sort of space station. The closest correlation to a story arch is introduction of mechanics and innovative uses of those build a difficulty curve and fun experience through the course of the game.

V. Immediate and long term projected socio/cultural project impact

The immediate impact is that people have yet another quick online game to play around with. In the long term it will hopefully help bring puzzle games back to the forefront of peoples' minds instead of FPS games or the like.

VI. Delivery System & Requirements

Have Java Runtime Environment enabled.

VII. Interface

On the screen is nothing but the level and your guy. Controls are introduced through the opening levels using special backgrounds with instructions. Keyboard keys control the character and the abilities.

The Level Editor interface will shrink the level enough to fit tiles and options at the bottom and side of the screen. Click on a tile to select the type you would like to place, then click on the corresponding grid, using the same rotation keys to rotate the piece that are used to rotate the room in-game. Save, Load, Test and Exit options appear at the bottom. It should be noted that while the level editor is mostly functional for our own level design purposes, it is extra in terms of the game itself, and was not able to be added for the symposium section.

VIII. User Interaction

Arrow keys control the character, Spacebar jumps, and A, S, and D rotate the room counterclockwise, 180, and clockwise respectively; it helps to think that the left-hand letter (‘A’) makes the left side of the room rotate down, and the right-hand letter (‘D’) rotates the right side downward. In the Level Editor, the mouse is used for selecting and placing objects, and A, S, D are used to rotate the objects. Other controls that have been added are M to toggle the music on and off, R restarts the level, and K skips the level. It is still up for debate whether the final version will allow skipping. If we had more time we would like to include a system for skipping, that would require you to earn a skip by winning X number of levels.

IX. The World Layout

The game takes you level by level. Each level is contained in a square, with an entrance and an exit door to signify the beginning and end of the level. Contained in each level is a challenge or a series of challenges to be overcome by the player in order to make it to the exit.

X. Level Design

Levels start out incredibly easy, and build up to introduce to the player what they can do. Each time a new mechanic is introduced, there should be a level devoted to teaching that mechanic, perhaps integrating it with some old ones to match the difficulty rank of where in the game the mechanic is introduced. Any other level could use any mechanic the player is familiar with to create a challenging environment and reach the finish. We have also included levels devoted to teaching players how to use moves, that will be used in later levels.

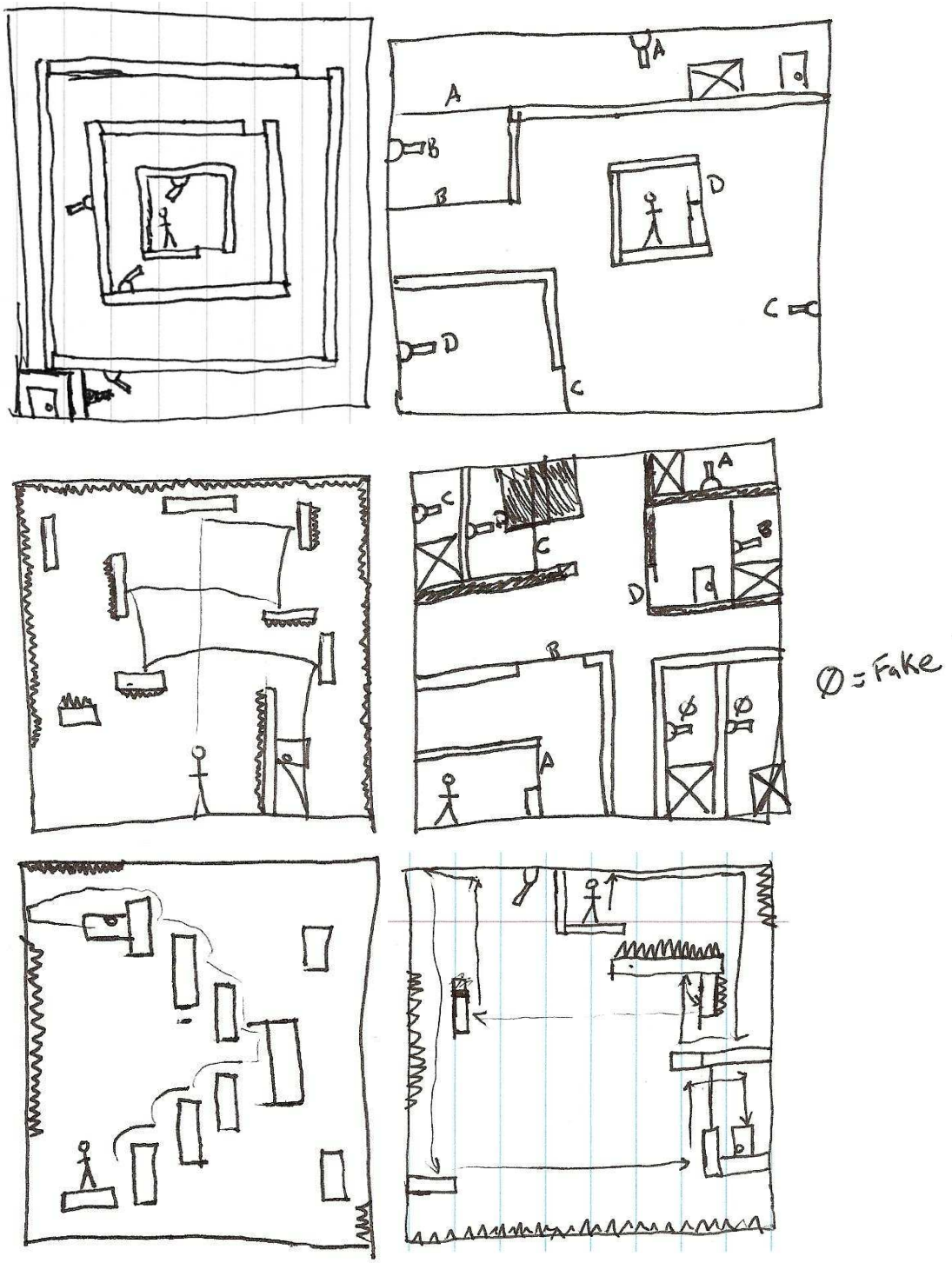
As we played and designed levels we became what you might call “expert gamers” with respect to our game. So even some of our simple levels proved to be a little challenging for some people, and what we considered moderately hard, was kind of insane for our play testers. Also through play testing we found that some things that we as designers noticed and worked around, other people would continually die on. We have changed our levels to make them more user friendly, and also to help players learn as we did. We found that because our game is a puzzle solver primarily and a plat former secondarily it was better to use lasers to make levels that would other wise to easy harder, though you can also use lasers to make a difficult level even more difficult. The trick is to restrict the player from being able to just get to the exit directly, but leave enough room for the player to play with our momentum mechanics. Through play testing we were able to determine what levels were more difficult than we intended, and we were able to adjust them.

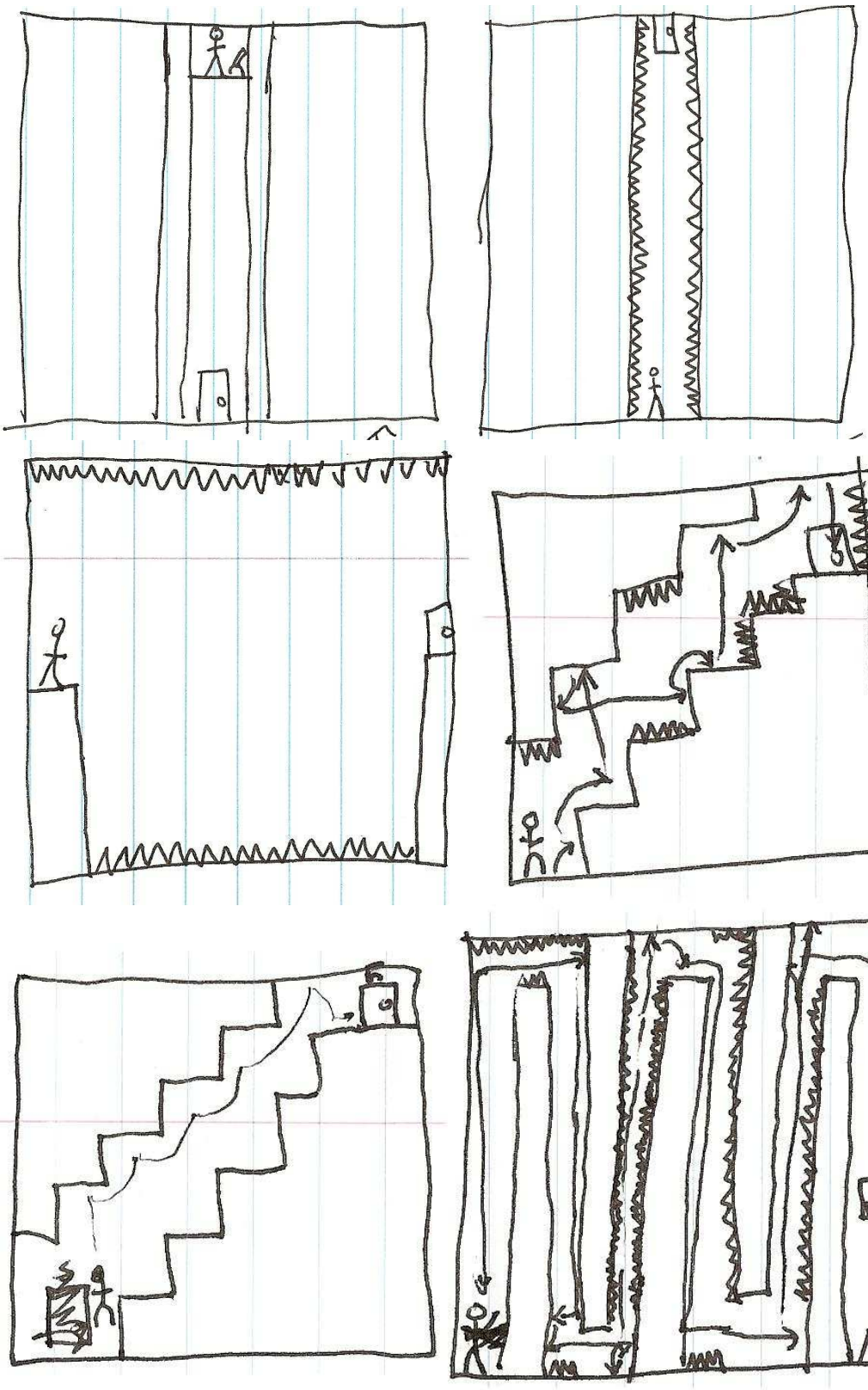
To sum up this lesson: If you think your game is too easy, make it easier. This of course depends on who you are marketing it to, but it’s a generally good rule of thumb. In the end, we were very proud that our game played like a true puzzle game, with most of the

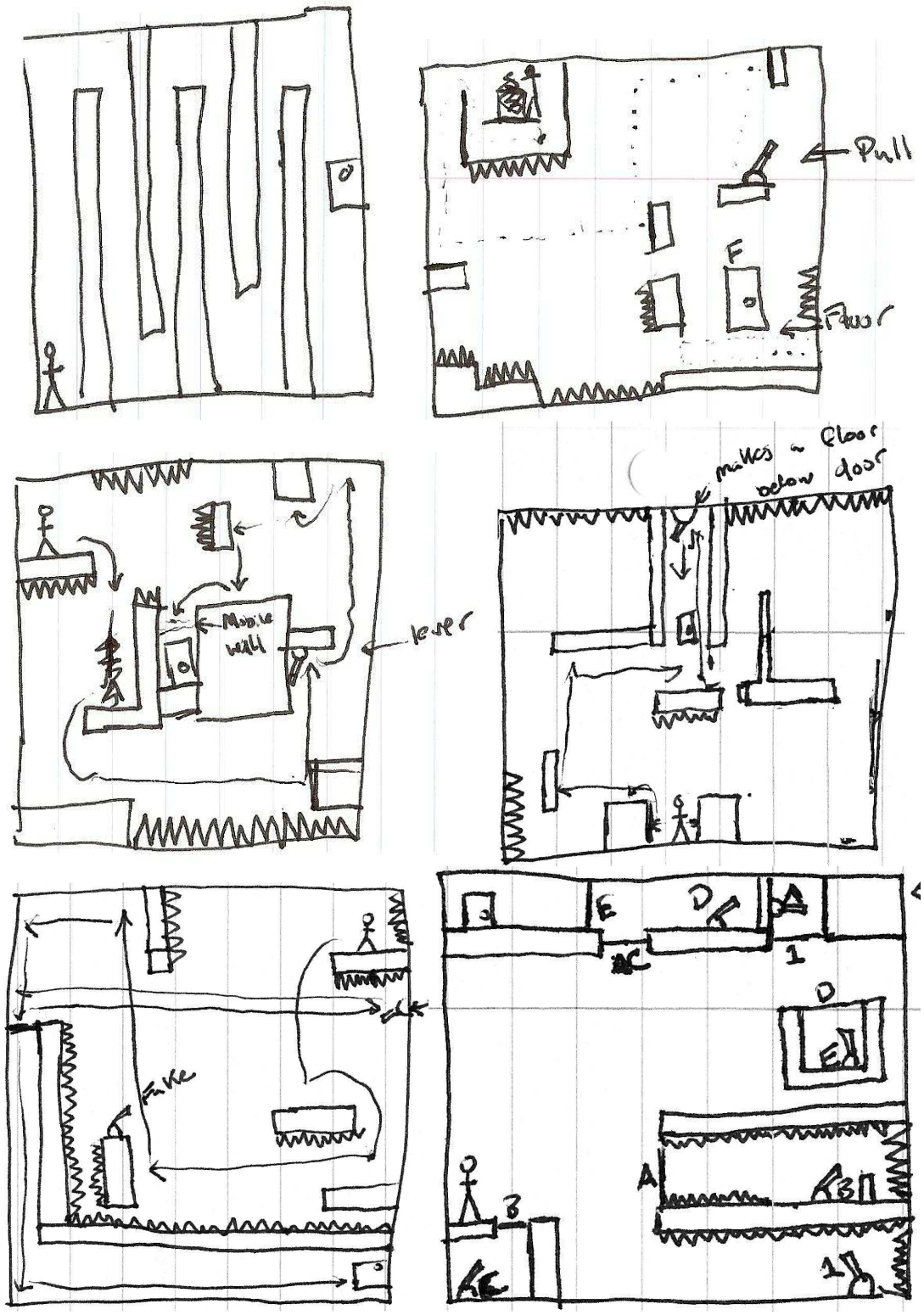
puzzles forgoing the necessity of precise timing and skill, and relying on the mind to find a simpler route to victory.

For a list of the mechanics, see Section XIII.

Here are some of the original level design sketches to get initial concepts on paper.





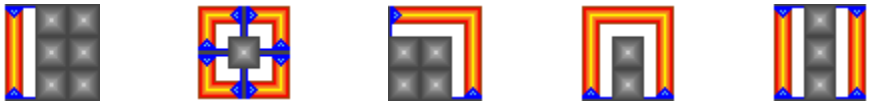


There are many level designs that were created in the level editor and didn't make it into the game; we also have many that did. But we won't show them all here.

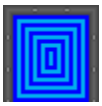
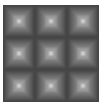
XI. Visualization- characters, flow charts,

Being in a space station, all objects have to seem somewhat sleek and space-like. Here are the tile pieces how they were originally designed.

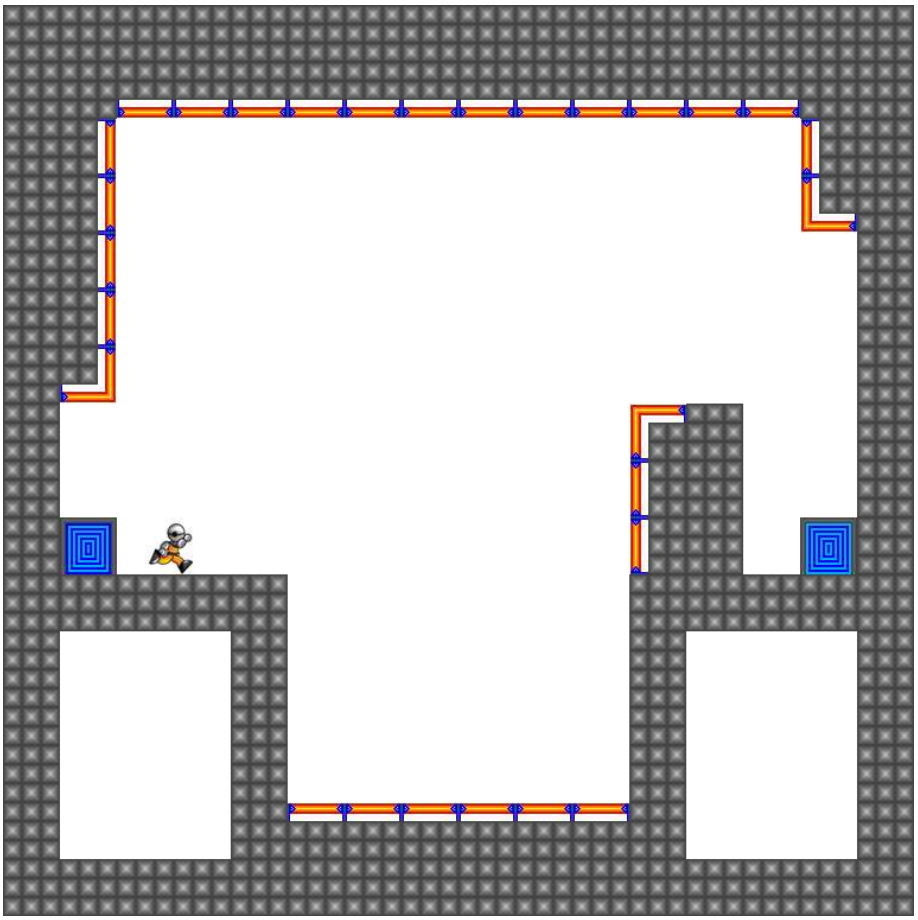
Lasers:



Other:

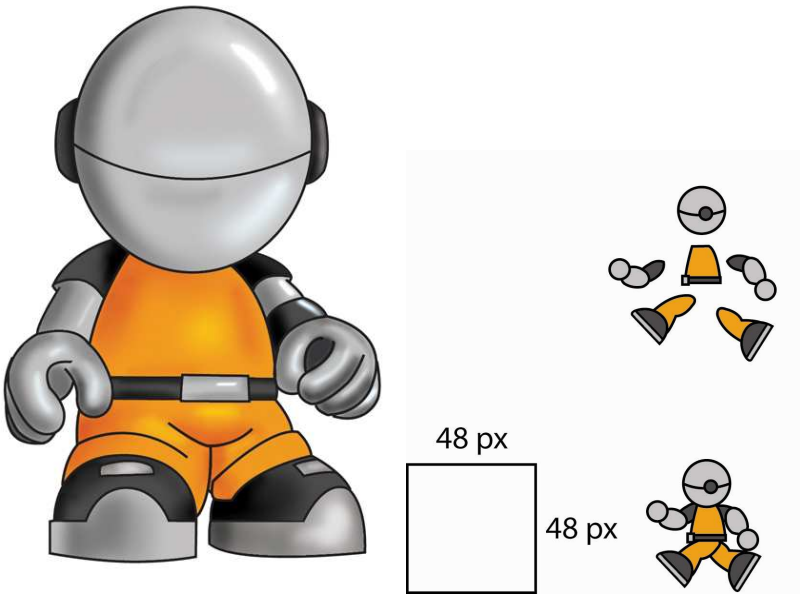
Crate  Door  Wall 

By combining all these elements, a typical level would look something like this:



As can be seen in the finally game, all of those underwent changes, in particular the wall sprite [Rory, this is your cue].

Here is the concept art for our character, and accompanying sprite versions. [Chris…]



XII. Music/ Sound Design

Music should be catchy, but not intrusive to the gameplay in any way; let the player think. [Rory again]

Sounds were carefully selected from the corners of the internet, using free and open resources only. We wanted spacey sounds that reflected the environment and the character. Sounds added were sounds for: opening a gate, closing a gate, rotating the room, landing on a laser, entering a level and landing after a rotation.

XIII. Rules and Gameplay A. Setup, B. Gameplay, C. Scoring

A. Rules
    a. Start wherever the map places you, and somehow make your way to the exit.
    b. Use the mechanics at your disposal in any way you see fit to make it to the exit. The following are all the mechanics that will appear in the game.
        i. **Primary**: Rotate the room 90 degrees CW, CCW, or complete 180.
        This is the entire purpose of the game. Every single level beyond maybe the first couple should involve this mechanic somehow.
        ii. Jumping

iii. Lasers – lasers travel along walls through nodes that contain them. They function the same way that most games employ spikes: if you touch a laser you will die and restart the level.

iv. Boxes - like the player, these are free objects subject to any movement and gravity. Fling them around using the rotation just like you would yourself. Stack them, stand on them, and most importantly use them to activate switches. They also are immune to lasers.

v. Switches and Gates

1. Switches act like a light switch, "attached" to the background of the level, when an object collides with it in the direction it is facing, they will flip, changing the status of all associated gates. Gates are linked through an "Object Subset" number possessed by all environmental objects, but utilized only by gates and switches. There are two kinds of switches; they both act the same, except that one kind only finds one subset number gates (0, 1, 2 or 3) and the other finds two different subsets of gates (0+1, 1+3, etc.) causing overlap with the other gates, and some interesting puzzle potential.

2. Gates start at the given location and extend until they encounter a space that is not blank. When a switch activates the gate, it opens, returning all the overridden spaces blank. If it is activated again, it closes (extends) again, switching its status.

vi. Mechanics that did not make the game due to time constraints.

1. Swinging Doors – these would always swing down from their hinges, so they would block a tunnel or what-have-you based on which way the room is rotated.

2. Laser switches – switches that turn off and on lasers.

B. Gameplay – Being able to switch the source of gravity is fun, and though playing with momentum and gravity is fun, playtesting revealed that it is also challenging and needed some getting used to. There were many calls to do away with it, but we felt that would have made the game too simple and not as dynamically real, so instead we simplified it and slowed it down.

C. Scoring – It is a thinking game, but that said there is something to be said for doing it in a short amount of time. Each game is timed, but we were not able to add the elapsed time into the game since it was extra and not central to gameplay. Instead, it prints out in the DOS window after each level.

With much more extra time, other things we might have added would be stats, so people could compete after they had already beat the game. Similar to Portal like number of steps, number of rotations, time and deaths.

XIV. Program Structure

The Game class runs the entire game, making sure the playable area, the level editor, and menus are only active one at a time, and painting them as necessary. It also holds functions to run the main game, and is the main panel in which the game appears. Game also runs the animations, so it stops everything from happening until an animation is complete; this includes entering, exiting, dying, and rotating.

The program has been constructed with two basic kinds of objects: Environmental and Free. Environmental objects are objects that get their own space on the grid that serves a purpose in the environment, such as a wall, or spikes, or a switch or a gate; they are not subject to gravity and maintain their grid position at all times. Whenever a free object collides with a grid cell, the colliding object is sent to the grid's object, figures out the direction of collision, and performs whatever action is defined for that kind of object (Walls, for instance, keep the free object on the side of the collision, not letting it through the grid).

Free objects, on the other hand, are subject to gravity and collisions. These refer to the Player and boxes. The program was initially designed with the intention of easily adding more if necessary, but shortcuts were taken to make boxes work, so some rewriting would be in order in that case, primarily in terms of saving and loading them. Free objects should also be able to collide with each other, so that boxes can stack or players can jump on them. There are still obvious "bugs" in terms of programming this, which still confuse me but don't affect gameplay enough for me to truly worry. Collision is tough, so don't be afraid to research ways to do it like I didn't.

These two classes allows for creating new categories for each with relative ease, allowing the game to evolve as more programmers get a hold of it. The grid system also allows for a level creator, simplifying the level design process immensely.

Biggest lesson learned through programming: plan everything out. Think about everything in the game you would need to program, and plan it out, even if you think it'll be easy to add. I planned out the Environmental Object class, the grid system, and how everything is saved, so now it is very easy to add another kind of object (we considered adding windows into space, but thought of it too late to bother adding) and is very easy to make levels. What I did not plan out was much of what would appear on the screen, namely animations; I also did not think through saving boxes. As a result, corners were cut in order to get this done, and lots of methods became extraneous from moving around variables and actions from different classes so in the end it became confusing and inefficient to read.

I also did not fully understand the concept of static variables. If I had, it would have made constructors much simpler for me, and I would not have had to reference everything to everything else – just make things like the Player or Level static variables in the Game class so they can be more easily accessed.

XV. Technical Specs: Physics, Rendering System, Lighting Models

The physics applied include gravity, momentum, and very little friction. Rendering and Lighting is irrelevant, as are most other physics concepts. I repeat, collision is hard to get working perfectly.

XVI. Implementation

Code generated from scratch using java and the given java library. Graphics created in Adobe Photoshop.

XVII. Production Timeframe

Biggest Scheduling Lesson: Plan an optimistic, ambitious schedule – plan to finish long before it's due – and stick to it. Even if you fall behind, don't rely on that extra time you set aside at the end; the more you fall behind the more and more late nights you will have as the deadline draws near. Pushing yourself early means things get done earlier, and problems are solved earlier to make room to solve newer problems. If in fact you finish earlier than planned, then use the extra time to either relax or polish the game to make it look and feel extra extra shiny and amazing.

- As of March 26th…
    - The Level editor has been completed from a programming standpoint.
    - All mechanics are working with the exception of free-object collision.
    - Several levels have been created.
    - Character run animation has been created.
    - Walls and lasers have been created.
    - Regular Game code is not in place.
    - Menu Screen graphic is in production.
- Left to do:
    - Add all graphics and sound as they come.
    - Complete all levels.
    - Complete main-game and main menu code
    - Create character-jump sprite/animation and character dying animation
    - Background(s)
    - Switch and Gate sprites.
    - Sound – Music and effects.
    - In-Game "pause" menu
    - Playtest.
    - Add in extras
        - Door animation
        - Laser-pulse animation
        - Level Editor

Schedule of remaining tasks to be done by.

- April 2nd

    - Add all available graphics.
    - Fix free-body on free-body collision.
    - Create switch and gate sprites.
    - Find key sounds, and preferably some music.
    - Complete Main Menu screen.
    - Start Enter/Exit level animations.
    - Plan all remaining levels, and difficulty arch of the game.
- April 9th
    - Fix bugs.
    - Add pause-menu code.
    - All levels should be completed.
    - Complete character jump sprite (turning it into animation is less of a priority, and can be done later).
    - Complete Enter/Exit animations
    - Complete sound and music search and add to game.
    - Complete background and all variations.
    - Start playtesting – send to everyone we know for feedback.
- April 16th
    - Fix bugs found in playtesting.
    - Fine tune the levels and difficulty.
    - Make Pause Menu image
    - Complete Character-dying-to-lasers animation
    - Catch up on anything not done. If all done, start on extras.
- April 23rd
    - Fine tune the game based on playtesting (this includes walking speed, jump height, maximum falling velocity, etc.).
    - Fix all remaining bugs.
    - Print all work and submit it all on a CD.

XVIII. Research

See Section II.

XIX. References

Shift – http://armorgames.com/play/751/shift
Portal – http://orange.half-life2.com/portal.html
And Yet It Moves - http://www.andyetitmoves.at/
Spin the Black Circle – http://www.bubblebox.com/play/skill/902.htm

XX. Other Lessons:

Always listen to the programmer. Of course it depends on the nature of the game, but more likely than not the programmer knows what is possible because without the

programmer there is no game (you can have a game without pretty graphics). This is not to demean the work of game artists out there, but is more to bring things down to a realistic and doable level. If the programmer can't make your game, then you don't have a game.
Also, more than one programmer is nice. It is a lot of work for just one person (again depending on the game).

When choosing a game, especially at the student level, pick something that will be easy for your programmer to program. This is similar to the previous lesson, but it is more a warning to the programmer. Things will come up that you will not think of. Even if everything is easy to program, it still takes time, and there are a lot of things that are taken for granted in games that the programmer is now expected to make happen (soft jumping for instance, and accurate and playable physics). Pick something easy to do, and – as with the optimistic schedule – if it is finished with time remaining, it can always play better and have more efficient code.

Playtesting is important!

Let me say that again.

Playesting is important!
Fortunately, this lesson had been pounded into us through others, and so we made sure to implement it. As explained in Section X (Level Design), the game had been constructed as much much harder than it appeared to us, hence the lesson "If it seems too easy, make it easier." Beyond difficulty, find out what players think of the game, what they like, what they don't, and strongly consider opinions that come up more than a couple times. Generally, this experience and feedback was incredibly valuable to us, and we took into account much of what was said. The hard part is getting people to say more than just "it was fun" or "it was frustrating."
One instance where we went slightly against the playtesters is on the issue of momentum. Many players could not grasp the fact that if you had some speed then moved gravity to another place that you continued to travel in the direction you had been traveling in instead of resetting your velocity. It was suggested by many to eliminate the concept entirely, however if that were done then the game would become incredibly easy, and levels would have to be added requiring accurate timing in order to make it more challenging – the game would have essentially become a puzzle-version of Pac Man. Besides, when you weren't dying, flying around made it more fun and feel real. The decision was made to keep momentum, but dumb it down a lot. The falling speed was reduced to ¾ of it's original, the maximum side-speed was set to half that of the vertical (so that when the room was "rotated" the player wasn't sent *shooting* sideways, but just enough to make it challenging and feel real), and the mid-air controls were given more sensitivity so it became easier to control. After this was done the game became much less frustrating while keeping the same feel.
For future reference, if the game were to be remade, a consideration should be actually changing the source of gravity instead of rotating the room. That way momentum feels

more fluid and in place, although you lose the cool affect of rotating the room and it would be harder to control your character.