The

Young Videogame Developer’s Journal

Ages 12-22 on the virtual frontier

By Chris DeLeon

www.deleongames.com

Table of Contents

KEY For Publisher or Purpose
IND Independent: Made entirely by, and for, myself, from an unassigned, unrewarded, unwavering, and unmistakable need to make videogames.
GCS Game Creation Society: Made with members of a college organization that I cofounded. See: **www.gamecreation.org**
SPE Sigma Phi Epsilon booth: Made as part of my fraternity's contributions to an annual 3-day public carnival at Carnegie Mellon University.
ECP/HCP/ICP English Class Project; History Class Project; Interface Class Project: Made for a class, generally as some type of requirements overkill.
ISF International Science Fair: Made for an internationally-ranked science fair project. My state ranked CS project/program isn't detailed in this text.
NIP Norwegian International Pair: Made as part of a two-person overseas team. Time zones are brutal!
3DR 3Dreams Demo: Made as an early demo for the 3Dreams engine by Raphael Mun.

Preface

Through a novel: imagine what I imagine.
Through a painting: see what I imagine.
Through a movie: witness what I imagine.
Through a videogame: explore what I imagine.

When I was still too young to handle a d-pad, I learned that – using both hands – I could press a light gun against the screen, flash it with the trigger, and affect ducks on the TV. What happened on the TV was up to me! Plainly, magic was behind all this.

And for ten years after that, I genuinely believed it was magic. Videogames came from overseas – "Japan" – where magicians mashed drawings, ideas, and music into plastic boxes for me to buy. What were the cartridges like on the inside? How could new worlds be created?

In late elementary, I made a starting realization, followed by a simple deduction:

A) Videogames are made by people.

B) I am a person.

A + B) That must mean that I, too, can make videogames.

I made up my mind to do just that. Not knowing where to start, I wandered into the local bookstore. There I saw *C for Dummies* by Dan Gookin. "I make mostly A's and B's on my schoolwork," I thought. "I'll start small, and read about C."

A little over ten years later, I am now working as a Technical Game Designer with Electronic Arts in sunny Los Angeles. Prior to my professional work, I served as a lead – lead designer, lead programmer, producer, or working solo – on somewhere around 40 independent games. As a cofounder and former manager of two years for the Game Creation Society, a college freeware videogame publisher, I was further responsible in varying degrees for the planning, milestone tracking, triaging, and continued promotion for another 15-35 videogames.

What happened during those ten years? Although I maintained a balanced life outside of videogames – enjoying athletics, art, philosophy, school clubs and great friends – the rest of my waking hours were spent pouring every ounce of my passion, mental energy, artistic dreams and force of will into making videogames. The videogames weren't always amazing, but I would like to think they were always at least decent, and I was learning a lot along the way.

To me, making videogames was not about critics. It was not about sales, market timing, or trends. It was not about money. It was not even about getting a job.

At that time in my life, making videogames was not a commercial effort.

It was not an academic pursuit. My zeal had nothing to do with understanding videogames, challenging assumptions, or categorizing ideologies. It simply didn't.

Nor was this an artistic pursuit! I was not aiming to express myself, document the human struggle, defend underdogs, push the limits or redefine what we call a "videogame."

It was about making videogames. Pure and simple. I spent a decade figuring out what I wanted to do, what I could do, how to do it, and doing it. I did it constantly.

This e-book highlights what I have learned about videogame development along that journey. This is not a programming book, nor is it an art book. It is not a how-to book. This is a page-by-page, succinct presentation of nearly every game that I have worked on, followed by what I consider the most valuable takeaways from each particular development experience.

Whether you're a fellow videogame developer looking to leverage my experience, a videogame player eager to have a behind-the-scenes look, or a spectator curious to learn more about this wondrous medium, I am hopeful that you'll find what you're looking for herein.

Approximately half of each page is devoted to a screenshot and a detailed breakdown. This makes for an unconventional layout for a game design book, but then, what good are videogame design suggestions if they cannot be understood in context?

There is a more depth to videogame design than building levels and tuning variables. Because videogame design is context specific, relatively few authors have written about the higher level, conceptual side to videogames. Among the few that have attempted this feat, most have milked a single example to the point of pedantic over-analysis, or provided ungrounded examples for lofty and over-generalized statements. At best, some advice is based upon a handful of personal experiences in the commercial videogame industry – where the feedback and lessons learned are typically market driven, diluted by hype, affected by licenses, hidden beneath layers of office politics, and rewarded like a Hollywood summer blockbuster for shameless pandering.

Here, then, are 45 unique and market-free contexts explained, alongside relevant design notes coming directly from a primary contributor to the original development teams. If you wish to explore a particular concept further, or play a few of the games to learn more about the context behind suggestions made here, www.deleongames.com has free links for many of these games.

You're encouraged to skip around. Browse the takeaways first, or skim the screenshots to find a videogame that interests you. Start from the back and work your way forwards. Explore, play, and do whatever you'd like. I intend for this e-book to be more like a library than a reading list, and there is no reason to read it cover-to-cover, page-by-page, point-by-point.

After all, playing a videogame the way the designer intended is never quite as fun.

What follows accounts for the lion's share of my self-guided learning and world experience in sheer hours, energy, and devotion from 1996-2007. Enjoy!

-Chris DeLeon
July 27, 2007

1996

Gameplay Mods

Screenshot

[Not available]

What I Did New levels, new units, and new graphics within commercial games
Other Primary Contributors None
Publisher/Purpose Independent
Release Date July 1996
Development Time A few weeks (each)
Content Length Hours
Quality of Outcome ⯫
Description

Flamethrowers and levels for **Doom** (id Software 1993) – Created a few types of flamethrowers by modifying existing weapons, and built my house as a map.

Jetpack for **Doom 2** (id Software 1994) – Modified the game to connect BFG firing to the code used to handle Archville attacks, resulting in a continuous jump.

New "portal" levels for **Descent** (Parallax Software 1994) – Took advantage of Descent data structures to build maps exploring inconsistent world geometry.

Electrical Bomb Launcher in **Command & Conquer** (Westwood Studios 1995) – Modified sprites for the SSM Honest John, and edited the game's unit balance to make the unit fair in single player.

Key Takeaways

- Content presentation matters. A lot. Changing the sprites or models can make a plasma weapon look like a fire weapon, and even if it "plays" exactly the same, it makes a huge difference to the player. Implications for the designer: people will pay more for a conceptually cool/threatening unit than a more conventional unit, even if it's technically not much more powerful, simply because it has better presence on the field. At a very real level, it can have a different psychological effect on how a human opponent perceives that unit. Damage and range statistics aside, for example, flame and chemical weapons are nastier than machineguns.
- Much of the game player experience happens in the imagination, not necessarily on-screen or in system memory.
- Just because something is possible in an engine doesn't make it a good idea. My experiments with making irrational portal levels (spatially impossible layouts) in **Descent** (Parallax Software 1994) were disorienting and not fun… which I rediscovered a decade later when I played **Prey** (Human Head Studios 2006).

Application of Learning

The lesson learned about the Electric Bomb's psychological value enabled me to boost the cost of the Crimson Army Mech in **Trichromic** (2007) beyond its statistical worth. Players still shell out an insane amount of money for it, and it is distractingly intimidating to other human players. That's well worth the money.

ASCII Games

Screenshot

[Not applicable]

What I Did Solo projects
Other Primary Contributors None
Publisher/Purpose Independent
Release Date December 1996
Development Time A few days (each)
Content Length <10 minutes
Quality of Outcome ⯩

Description

Crude ASCII (text-based) games: **Craps**, **Battleship** (grid), **Snake**, **DeliDraw**.

Key Takeaways

- I didn't know anyone in my hometown with a Computer Science background, and the only CS in my public education would be a typing class 3 years later, so no one was there to say, "Start with BASIC." Lesson learned: never tell a kid what s/he is or isn't ready to learn.
- Programming in C/C++ isn't very hard. Especially while young enough to not know any better, and when you're working alone so that there's no fear of making a tangled mess from things as long as the project gets tied together.
- Text is limiting. It takes away simultaneous action, overlapping timers, the ability to present multiple pieces of state or world information in parallel, and the ability to provide content compulsions for the player.
- Image editing software and games are only differentiated by logic which restricts each to its respective functionality. Tools can exist within games, just as easily as games can exist within tools.
- Starting small pays off. I spent months writing text games before making the move to 2D per-pixel graphics, years doing 2D per-pixel graphics before moving to sprites, and years doing sprites before tinkering with 3D or vector-based rendering. In turn, I have developed a much deeper understanding of graphics, visual design, and effective development pipelines at a wealth of altitudes, suitable for a wide range of areas within a single game (HUD, menus, world…).
- Computers are remarkably powerful and unforgivably obedient. The only requirement to control them is that you have to "speak their language."

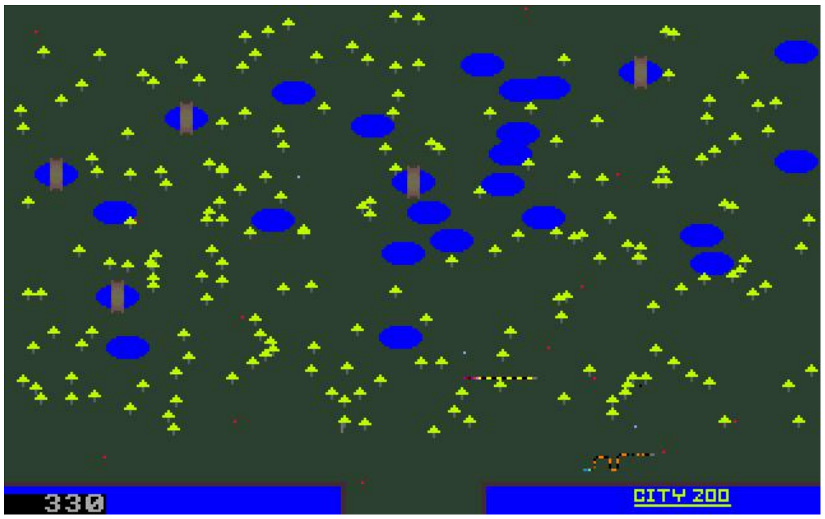
Application of Learning

My ASCII based game of **Snake** led to **Fugitive** (1997), which began as an experiment using pixels to render what I had previously been forcing to the screen as white on black characters. In the transition from ASCII to graphics I also made a handful of other incomplete, messy pixel based games, including a crude graphical game called **Tank** that eventually became **Pac-Deli** (1997), and a handful of bouncing particle demos that became **Sky Rake**'s (1997) engine. The **Battleship** game I did at this time is of absolutely no relation to **Battleship 88** (2005**).**

1997

Fugitive

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date March 1997
Development Time 3 weeks
Content Length 30 minutes
Quality of Outcome ★½
Description

Fugitive is a one or two player "snake" style game with power-ups (apples to freeze or blow up enemies) and simple hunters chasing the player. The player loses tail by moving through trees, grows tail by collecting apples, sprints by holding a direction, and dies instantly for touching water or a hunter. If either player dies, both players have to restart. The game ends after 40 apples.

Key Takeaways

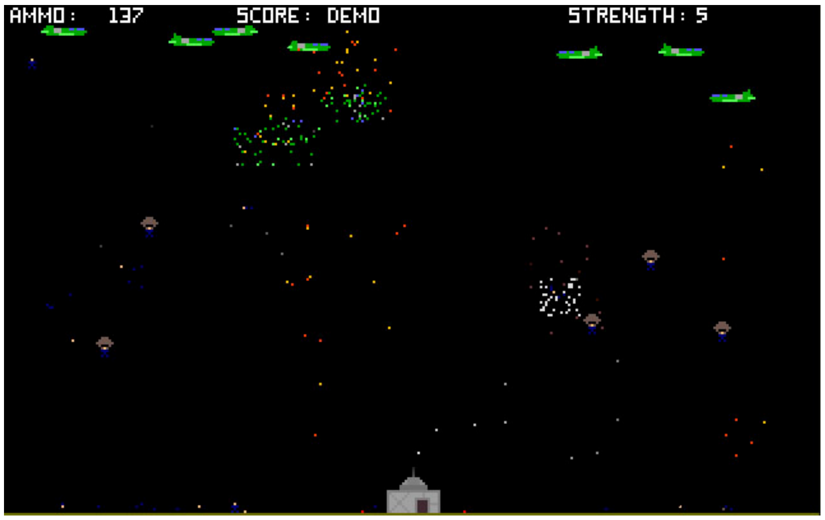
- An otherwise relatively meager game can be surprisingly fun in 2-player mode. This helped me understand why shallow arcade beat'em ups didn't appeal to me with single-player gameplay, even though I could play with another person for hours through the entire game. Simple gameplay keeps attention free for talking.
- Making both players accountable to the team's fate led to greater amount of teamwork in misleading/confusing the hunters to save one another.
- Randomly generated levels suffice when a game is focused on object interactions.
- Layout of nature is random; layout of manmade work is orderly.
- Having an in-game manual makes the game more accessible to new players.

Application of Learning

I experimented with lop-sided variations on teammate accountability within a single player context with **911** (1998), and then again in **GoD 2: Guardian of Dawn** (2003). **Burn 2** (2001) utilized random level generation.

Sky Rake

Screenshot



What I Did Solo project
Other Primary Contributors PLBM Games (design inspiration)
Publisher/Purpose Independent
Release Date May 1997
Development Time 2 weeks
Content Length 1 hour
Quality of Outcome ★★
Description

A re-interpretation of the classic **Ack-Ack Attack** by PLBM Games. Sky Rake features special modes unlockable by high scores, a demo mode played in real-time by AI, protective landmines, and various enemy types. Unlockable cheats include SAMs, Flak, alien enemies, 2 player mode, enemies without parachutes, etc. Enemy types include Demolitions, Spies, and EMP/stun bombs.

Key Takeaways

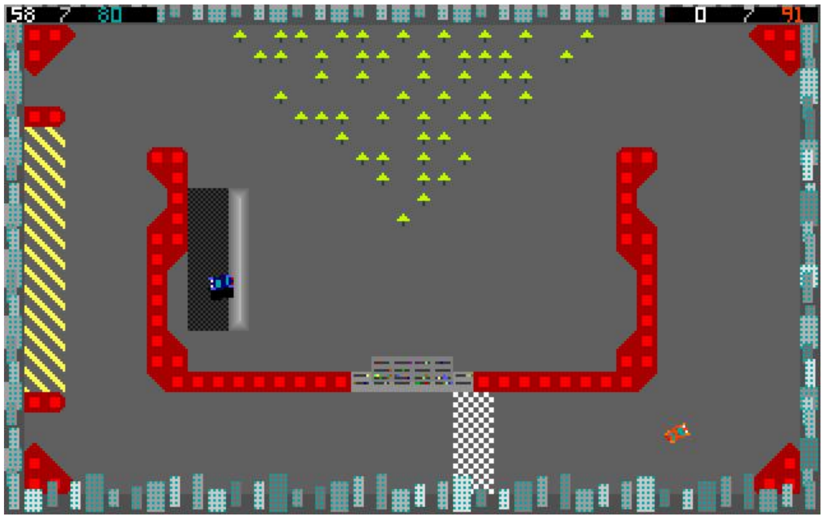
- Unlockable cheats do wonders for replayability.
- Players notice and appreciate fine attention to detail; I didn't expect anyone to notice bullets fading to black, destroyed objects decomposing pixel-for-pixel, enemies getting parachutes cut by shrapnel and falling to their death, etc.
- Training mode wasn't worth the time and effort. Players learn best by playing.
- When particle effects seem a little excessive, they're probably just about right.
- Randomly generated fire effects are cheap at a per-pixel level in low resolution.
- Continuously influencing chaotic, disordered, interrelating systems is fun. Plane shrapnel blows up other planes, throwing out more shrapnel, cutting parachutes…

Application of Learning

Unlockable cheats were a motivator for **Swarm** (2000) and **Battleship 88** (2005), both with positive results from players. Randomly generated fire showed up in **911** (1998), **The Guinea Pig** (2003), **War of the Worlds** (1999), and **Battleship 88** (2005).

Road Rage

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date July 1997
Development Time 1 month
Content Length 30 minutes
Quality of Outcome ★
Description

A two-player overhead racing game with 10 race tracks, 10 destruction derby arenas, and 10 miscellaneous practice areas. The levels feature healing areas, spike areas, ramps, and walls/cities/trees (all of which behave the same). Cars control in fluid 360 degree turns. Car health starts at 100%, affects acceleration and top speed, and can rise up to 125% in the pit areas.

Key Takeaways

- Car damage deteriorating performance can be frustrating, but due to the small track sizes and the speed of healing, here it made for a rather workable dynamic.
- The cars were only 28 pixels, but since they rotated smoothly and drove fast, the simplistic avatar came across as perfectly convincing.
- This was my first game with deliberate “level design.” I was intrigued at how the art of level design was (A.) so game specific (B.) so unscientific and (C.) obvious when done poorly but a mere player afterthought when done well.

Application of Learning

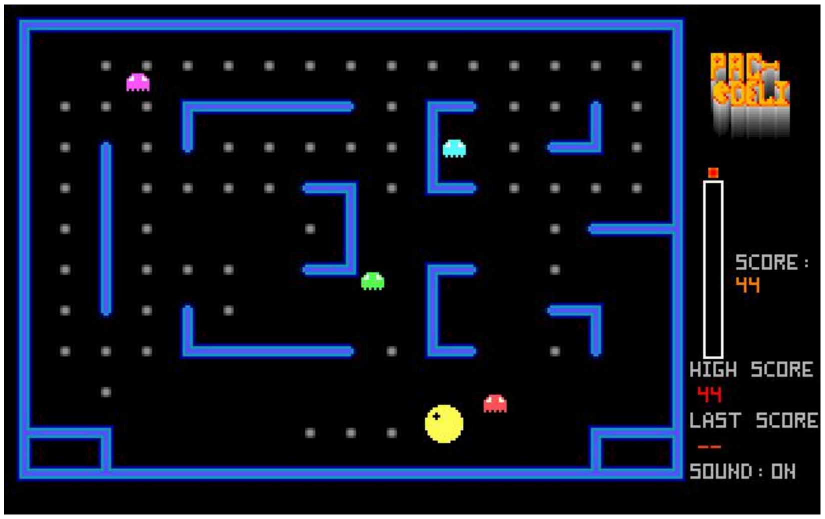
Guile of Dybbuk (2002), a first person shooter, also impaired player movement when hurt. Levels were larger, healing was rarer, and enemies didn’t face equivalent impairments, so this didn’t fit as well as it did in **Road Rage**.

A vertical offset with a black shadow became how I did vertical movement for **LeapFrog** (1998), **Rowdy Rollerz** (2004), **Battleship 88** (2005), and **Eternal Storm** (2006).

My interest in level design here led to my next project, **Pac-Deli** (1997).

Pac-Deli

Screenshot



What I Did Solo project
Other Primary Contributors Namco (characters, concept, inspiration)
Publisher/Purpose Independent
Release Date November 1997
Development Time 3 weeks
Content Length 45 minutes
Quality of Outcome ★★★
Description

It looked a little like Pac-Man, but it plays differently. The tunnels are no longer tight, giving Pac-Man and the ghosts more elbow room to move. That room helps emphasize differences in ghost moods/personalities.

There are 4 possible "moods": aggressively chase player, player mimic, aggressively chase another ghost, or wander. Every 1-5 seconds a ghost picks a new mood, keeping a bias for one mood in particular (its "personality").

The session's high score is displayed prominently on screen next to the last score reached. Eating ghosts is effective until the next level begins.

Key Takeaways

- The personalities came through quite clear, and really became an integral part of how the game was played.
- Screen transitions are easy to add, and make a game seem far more polished.
- The independent "intelligence" types created the impression of emergent "team-work," where casual players would blame a higher degree of coordination than was happening. "Purple distracted me while Blue moved in discretely for the kill." Very little complexity was needed to create the illusion of intelligence.

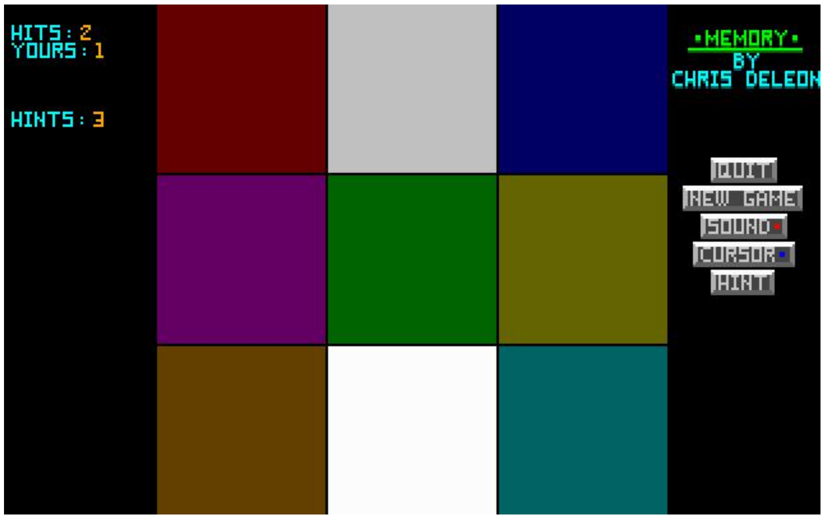
Application of Learning

LeapFrog (1998) further tested the emergent-intelligence hypothesis by introducing even simpler AI's (sans personalities) to simulate teamwork. I used "personalities" and "moods" again in **Swarm** (2000) AI, and I used similar AI tricks for **Qubestraphobic** (2005).

1998

Memory

Screenshot



What I Did Solo project
Other Primary Contributors Ralph Baer (inspiration)
Publisher/Purpose Independent
Release Date January 1998
Development Time Several hours
Content Length 10 minutes
Quality of Outcome ★
Description

Memory is a variation on the handheld game **Simon** by Ralph Baer. The game presents incrementally longer strings of lights and sounds. Color, spatial relationships, and audio frequency are distinct and consistent for each light. The player must click the sequence, in order, after each time the sequence plays. Three "Hints" (free next moves) are given at the start of each game.

Key Takeaways

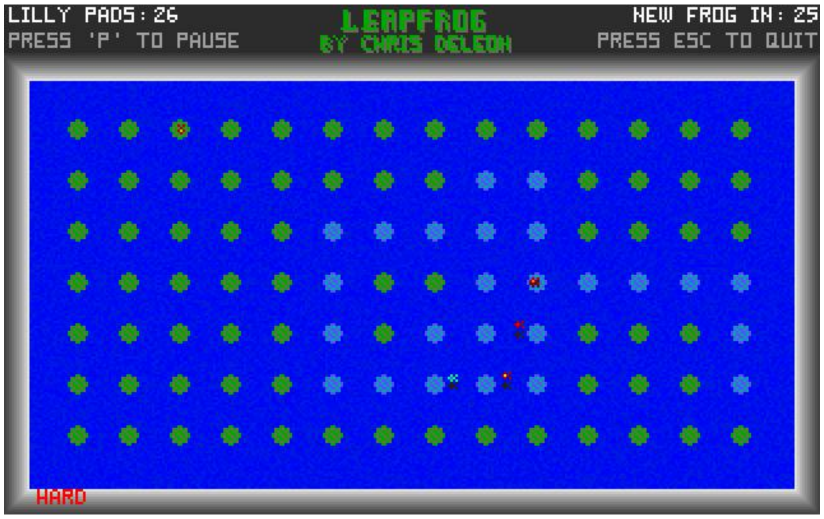
- Puzzle games can be done incredibly fast, and puzzle games have phenomenally simpler content requirements than their world-embodying cousins.
- In-game Reset options are important to prevent player frustration from bad starts.
- The amount of time and happiness that an interested player can derive from a game is not a function of its development time or complexity.
- Sounds, even simple tones, can be rewarding or punishing feedback to a player.
- A well designed mouse interface requires little or no explanation to the user.
- 30 years after **Simon**, this game is still enjoyable. Well-conceived fun is timeless.
- Programming isn't hard. Game design – deciding *what* to program – is the trick.

Application of Learning

Cacophonic sound alerting the player of incorrect behavior was used to condition players to avoid fire in **911** (1998), and later to motivate racers in the PC port of **Super Push 64** (2007) to avoid the edge of the course.

LeapFrog

Screenshot



What I Did Solo project
Other Primary Contributors Warren Davis and Jeff Lee (design inspiration)
Publisher/Purpose Independent
Release Date March 1998
Development Time 2 weeks
Content Length 30 minutes
Quality of Outcome ★★½
Description

The gameplay works conceptually like **Q-Bert** (Davis and Lee 1982): discrete movements, emphasizing timing to avoid enemy movements, are used to "mark" every level unit by touching it with your feet. Touched pads turn blue. Although blue pads are still usable to maneuver, they serve no other benefit.

Leap Frog was designed around one question: Can independent, semi-randomly behaving simple entities appear to demonstrate teamwork?

Key Takeaways

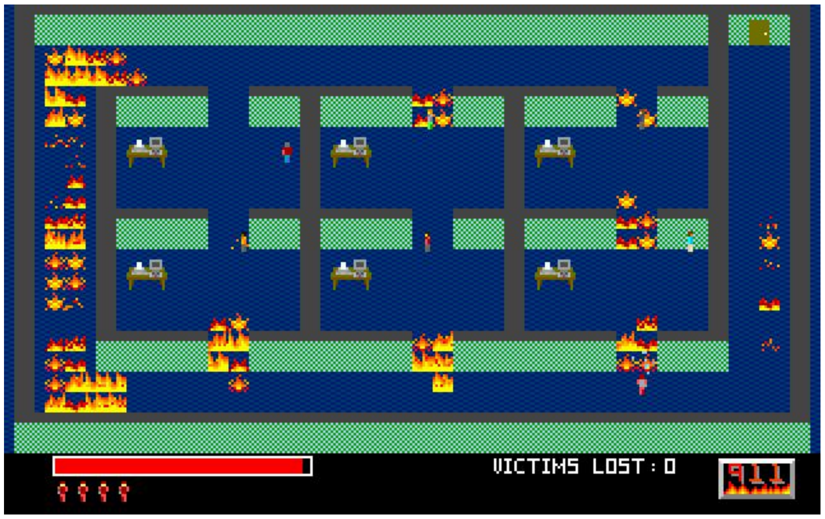
- Providing different difficulty modes both increased accessibility for a wider range of players and increased gratification for skilled players.
- Periodic path deviations, random hesitation, and otherwise aggressive behavior conveyed the impression that the AI was conspiring to corner the player. In a narrow enough context, with enough agents, semi-random behavior *is* intelligent.
- Enemies jumped at different times, making the player "juke" the enemy with sporadic, misleading movements. Unpredictability from the opponents, paired with short-term inevitability (the frogs move slowly through the air) raised the tension for the player by limiting foresight until it comes too late to be helpful.

Application of Learning

Difficulty modes found their way into **DelishPong** (1998). The seeming emergence of "coordinated teamwork" AI from random localized decisions helped me focus on per-unit behaviors in my next game, **911** (1998), and many of my games after that.

911

Screenshot



IMPORTANT CONTEXTUAL DISCLAIMER

This was made in 1998, 3 years before the "September 11" attacks. The title is pronounced "nine-one-one" for emergency phone calls, not "nine-eleven."

What I Did Solo project

Other Primary Contributors None

Publisher/Purpose Independent

Release Date May 1998

Development Time 3 weeks

Content Length 1 hour

Quality of Outcome ★⯨

Description

Fires spreads as probabilistic cellular automata, and the player must use a fire hose to put out all fires before the panicked civilians become trapped and die.

Key Takeaways

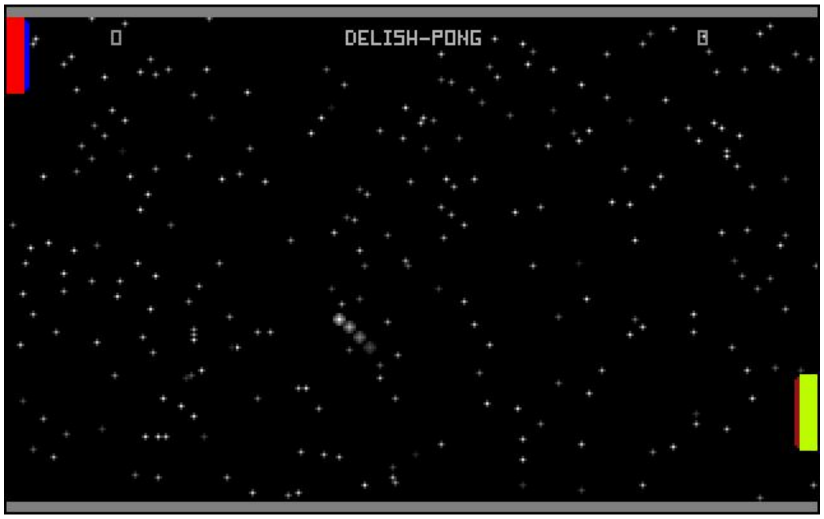
- The game was more interesting to me as a computer programmer than it was to the players - just because a game is original and works doesn't make it fun.
- Using 3 different animated icons to represent levels of fire severity was an effective way to rapidly convey the map tile's value of 0-255 heat to the player.
- Making the player's death less likely/problematic than the death of the panicked civilians made for an interesting game space coverage compulsion.
- Making a visual level editing tool part of the game's engine made levels far easier to develop nicely than was possible with hard-coding text grids.
- Hoses are complicated – so I gave the firefighter an infinite backpack of water, and no one ever said a word about it. Most players prefer cool over realistic.
- Giving multiple lives to the player is about sweeping designer-caused unfair deaths under the rug, not forgiving player mistakes.

Application of Learning

GoD 2: Guardian of Dawn (2003) explored the idea of a strong player losing when helpless AI's are defeated. I avoided lives until **BRAD** (2005).

DelishPong

Screenshot



What I Did Solo project
Other Primary Contributors Ralph Baer and Al Acorn (design inspiration)
Publisher/Purpose Independent
Release Date September 1998
Development Time 2 weeks
Content Length 10 minutes
Quality of Outcome ★⯨
Description

DelishPong is a game of *reflection* pong – not *classic* pong – because the paddles cannot affect the return angle of the ball. Upon hitting a paddle, the ball performs a vertical mirror reflection and accelerates by some percentage. The game can be played between people, or with 1 of 3 AI's on either side:

1. Beginner (Simpleton, Biff): slowly aligns with the ball's vertical position
2. Medium (Joe Blow, Brad Gower): returns paddle to center while ball moves away, and computes a single reflection of the ball on its way back
3. Hard (Professor Boskonovich, Professor Hamzaee): anticipates and moves to intercept the ball up to 3 reflections in advance

Key Takeaways

- Sparking and shooting stars made the background much livelier.
- Writing AI that could play against itself took the fatigue out of debugging it.
- Adding a trail to the ball made it easier to see and track at high speeds.
- Because the ball reflects in **DelishPong**, each player's anticipation is purely geometric instead of behavioral, and there is no reward for risky edge-blocks.
- For simple games: people are happy to playtest, and the feedback is priceless.
- Algebra really does have its uses – and math is far easier to learn in context.

Application of Learning

Shooting stars and AI's that played against AI's found their way into **Burn 2** (2001), and giving a fast ball a trail is used in **Ghosts in the Machine** (2007).

1999

Who Wants to be a Millionaire: Ancient History Edition

Screenshot

[Not available]

What I Did Solo project
Other Primary Contributors None
Publisher/Purpose History Class Project
Release Date February 1999
Development Time 1 month
Content Length 10 minutes
Quality of Outcome ⯨
Description

My teacher for Ancient and Medieval History required that we find an unconventional way of presenting a batch of 20 terms and their historical definitions. I made a **Who Wants to be a Millionaire** PC game with an avatar modeled to look like our teacher. The interface crudely resembled the television show, but nothing within the game was animated.

Key Takeaways

- Not only is it acceptable to turn in a videogame for a school project, but the teacher and students respond quite well to it.
- Quiz games are quick and easy to program, but the presentation layer becomes increasingly important since gameplay probably won't keep the game afloat.
- Loading quiz data from an external text file saved time, and it also allowed me to fix typos up to the last minute without recompiling.
- The soft coded data also made the distributed program flexible to work with new questions/content (not that anyone modded it…).
- Alternate answers, when picked from a bin of answers from other questions, were often obviously wrong matches. Even a multiple choice game needs "level design" insofar as alternatives must be carefully picked for each question.
- This was my first project using the Allegro[1] library to handle graphics, input, and audio. Before this, and for awhile after this, I used my own routines to plot pixels, change video modes, and produce sounds. Effective use of the Allegro library led to massive changes in how I created games over the next few years…

Application of Learning

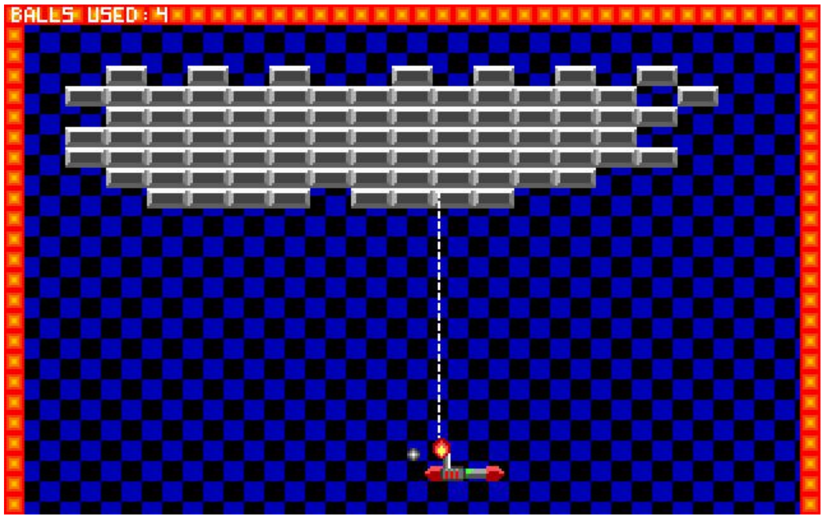
Years later I turned in a massive **War of the Worlds** (2002) game for an English class assignment.

Five years later I did another, more elaborate quiz game, **One of These Things is Not Like the Other** (2003), as a public kiosk game in my fraternity's Spring Carnival Booth.

[1] Developed by Shawn Hargreaves; See www.allegro.cc for more information.

B-Breaker

Screenshot



What I Did Solo project
Other Primary Contributors Nolan Bushnell and Steve Wozniak (design inspiration)
Publisher/Purpose Independent
Release Date August 1999
Development Time 1 month
Content Length 2 hours
Quality of Outcome ★★
Description

36 level **Breakout** (Wozniak 1976) semi-clone with three difficulties and powerups – gun, ball return, ball slow-down, random powerup, etc. Unlike the classic **Breakout**, the ball in **B-Breaker** will bounce off brick backs and edges, instead of carrying polarity from last touch with paddle or back edge. Also unlike **Breakout**, the ball does not speed up or earn different points based on the highest brick row touched, and there is not a penalty for touching the back edge.

Key Takeaways

- No matter what fancy layout I'd try to do with the brick and wall placement, everyone's favorite level was just the simple, level one, rectangle arrangement. If the gameplay is fundamentally fun, level design needs to not get in the way.
- Players didn't enjoy the "challenge" levels – people just wanted a way to relax. People play **Breakout** type games for a restful, Solitaire-type experience.
- In recognition of the player's want to simply play, I provided the player with infinite retries. That may be the smartest move I made in developing **B-Breaker**.
- The level format that I defined allowed me to cut gaps in the level edge barriers. Levels that had gaps (and there were a few) felt hopelessly frustrating. *Once again:* just because an engine supports a feature is not a good reason to do it.

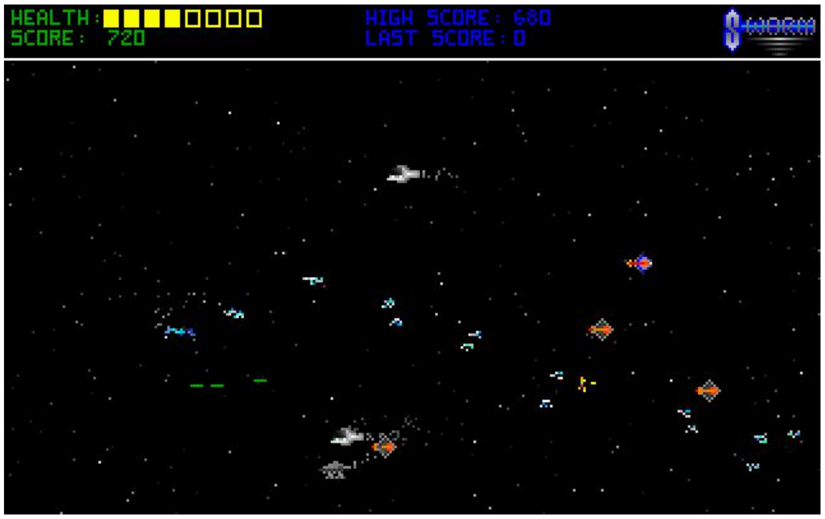
Application of Learning

In **Ghosts in the Machine** (2007) – also abstract and classically inspired – I kept backgrounds plain and dark. I learned from **B-Breaker** that needless visual clutter from busy, patterned backgrounds can detract from the gameplay elements.

2000

Swarm

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date January 2000
Development Time 6 weeks
Content Length 2 hours
Quality of Outcome ★★★½
Description

Swarm is 9 level side-scrolling space shooter with 3 player ships options:

1. USS Escher: Big, slow, with a machinegun firing through multiple targets
2. X-304 Raelynn: Mid-size, fires scattering pairs of lightning projectiles
3. USAF HighHawk: Tiny, agile, fragile, with heat-seeking rockets

The enemies field 5 types of units: 2 laser drone types, 2 bomber types, and cloaked interceptors. Enemy behavior revolved around personalities and moods (see **Pac-Deli** 1997), but were heavily influenced by player interaction.

Completing Level/Story Mode unlocked a different feature based on which ship the player used: all-3-ship-weapons-at-once (Escher), nuclear rockets (Raelynn), or a hidden ship with its own weapon, the Phantom (Highhawk).

Key Takeaways

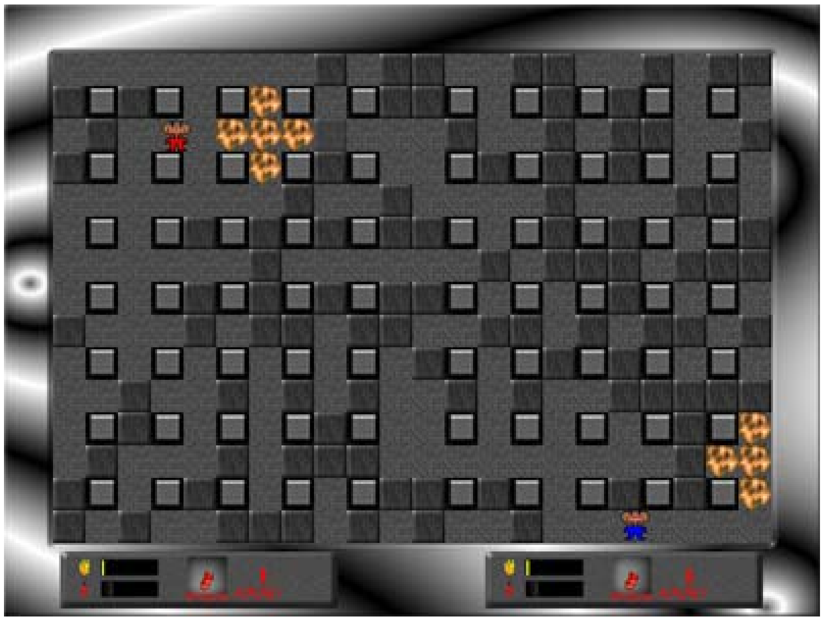
- The three player ships played quite differently, with unique advantages and strategies, greatly increasing reply value (furthered by unlocking unique cheats).
- The in-game manual provided background information on all enemy player and enemy ships, which players reported as building better depth into the game world.

Application of Learning

The X-304's spread shot pattern creates a conflict for the player between wanting to move in close, while wanting to keep distance for easier defensive maneuvering. This "spray" weapon mechanic found its way into **Burn 2** (2001).

Ka-Bom 2

Screenshot



*[Screenshot from original **Ka-Bom; Ka-Bom 2** image not available]*

What I Did Sprite Art, 3D Models/Animation, Game Design, Level Design
Other Primary Contributors "Zionic" (game logic, producer), Hudson Soft (concept)
Publisher/Purpose Norwegian partner project
Release Date N/A (Lost/abandoned in August 2000)
Development Time 5 weeks
Content Length 1 hour
Quality of Outcome ★✓
Description

A developer in Norway, Zionic, created a Bomberman clone called **Ka-Bom** using Game Factory. We met online. It became clear that he wanted to do a sequel, so I offered to help 3D model/render character sprites (3D animated soldier), conceive of new powerups (hand grenades, C4…), and build new levels for **Ka-Bom 2**. 70% through the project's development, Zionic informed me that he lost the game's project files, and that he had no backups of anything. Without anywhere to turn, **Ka-Bom 2** consequently vanished into the æther.

Key Takeaways

- Back up your project development data. Frequently. Externally. All of it. Now.
- It's difficult, and in some cases impossible, to hold an overseas or otherwise remote colleague accountable for a project's outcome.

Application of Learning

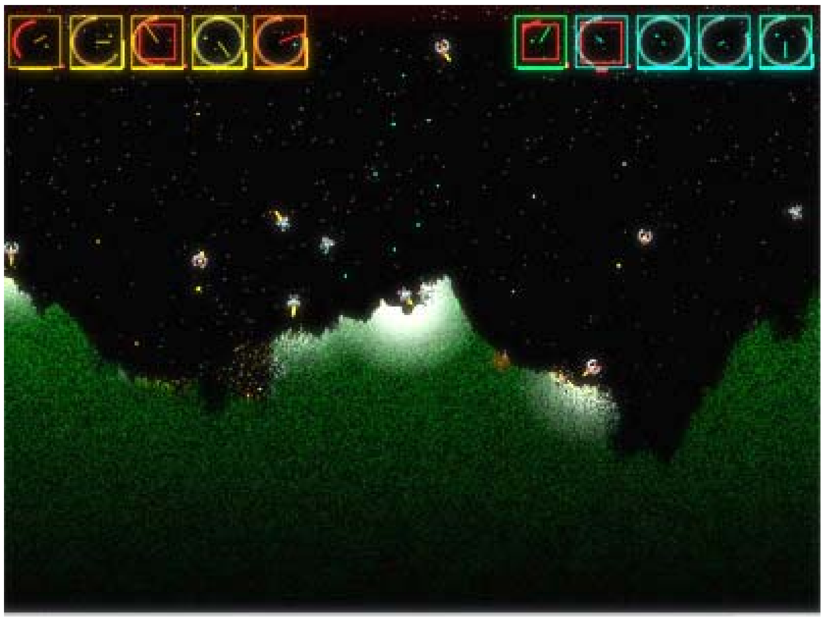
When a hard drive failure destroyed my latest development version of **Burn 2** (2001), my newfound habit of backing up projects weekly paid off.

After working alone for years, I realized the main error I made was in working with someone that I couldn't hold accountable. I did my best to account for this project management lesson when co-founding the GCS in 2004.

2001

Burn 2

Screenshot



What I Did Solo project
Other Primary Contributors Rich Adam and Mike Hally, of **Gravitar,** for inspiration
Publisher/Purpose Independent
Release Date May 2001
Development Time 2 months
Content Length 3 hours
Quality of Outcome ★★★★★
Description

Burn 2 is a team deathmatch variation on the classic **Gravitar** (1982), with teams up to 5-on-5, as many as 5 simultaneous human players, randomly-generated (often inhabited!) planetscapes, and fully destructible terrain.

Key Takeaways

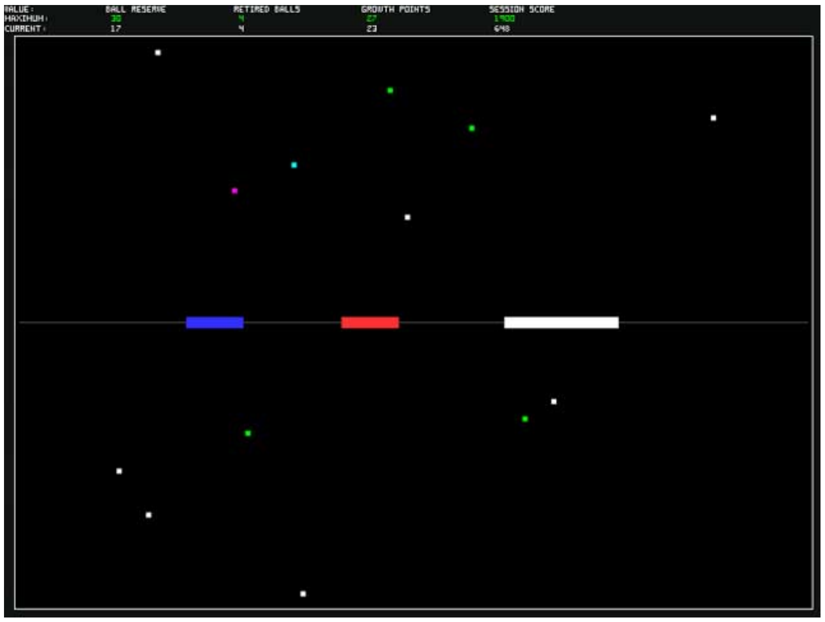
- High speed arcade-style action games leads to heuristic AI development, since there's no ideal behavior; the skill comes from balancing dynamic strategies.
- Custom data structures and efficient programming can allow for particularly unique features, like 2D destructible worlds, and custom colored lighting effects.
- For a game that's about the fundamental interactions of the units in play, rather than the play space, procedurally generating a playing field was the best solution.

Application of Learning

Burn 2 takes place via constant adjustments to heading and speed, counter-balancing momentum while aiming, and timing attacks dependent upon relative positions and speeds of targets. **Burn 2** is the spiritual predecessor to my most successful work, **Battleship 88** (2005). The fun that I had with **Burn 2**'s heuristic AI stole away my attention again for **Ghosts in the Machine** (2007).

Gehdiun

Screenshot



VALUE:	BALL RESERVE	RETIRED BALLS	GROWTH POINTS	SESSION SCORE
MAXIMUM:	30	4	27	1900
CURRENT:	17	4	23	648

What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date October 2001
Development Time 1 day
Content Length 10 minutes
Quality of Outcome ★★
Description

Three paddles: White, Red, and Blue. The mouse moves the white paddle left or right. Left Clicking repositions the Red paddle, Right Clicking repositions the Blue paddle. The game starts with one ball on the top half, and each time a paddle prevents crossing, it speeds up and advances to the next of 7 colors. The objective is to juggle an increasing number of balls without letting them past the center line. Scoring was based on the number of "fully juggled" (at least 7 hits) balls present at the time of each deflection.

Key Takeaways

- I enjoy it **Gehdiun**. No one else seems to. It was worth 1 evening of work/
- Crazy experiments like this are best kept out of huge core projects, where more of my life, effort, and money are at stake for failure.
- Sound and music are very important. Abstract play doesn't change that, and this game definitely suffered for the loss.

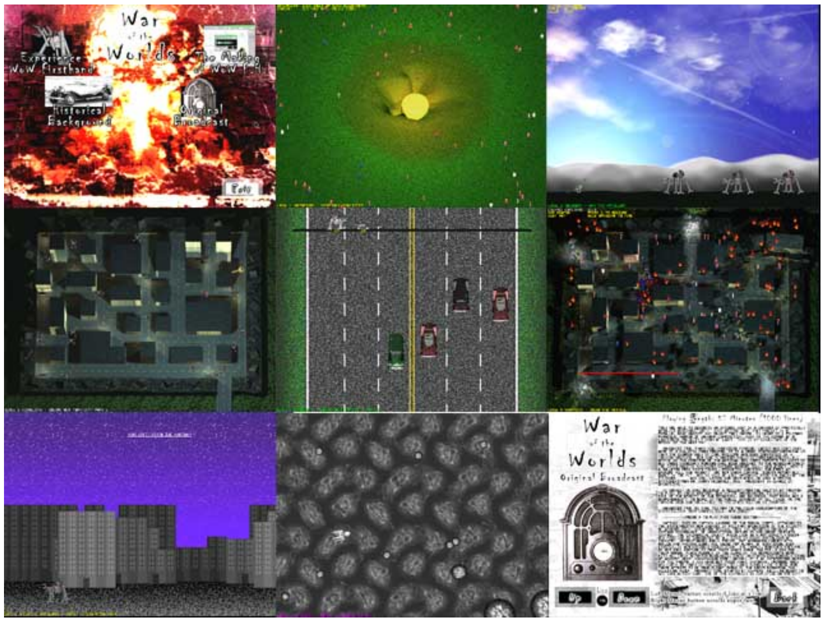
Application of Learning

Ghosts in the Machine (2007) is the a spiritual successor to **Gehdiun**, in that it represents an unconventional approach to input mechanics as a modification to a classical gameplay model – **Warlords** (Atari 1980) for **Ghosts** instead of **Pong** (Atari 1970) as was the basis for **Gehdiun**.

2002

War of the Worlds

Screenshot



What I Did Everything except the concrete poems, gameplay concept for 3 levels, and IP
Other Primary Contributors Shil Patel (history details), Allison Morrow (poems)
Publisher/Purpose English Class Project
Release Date April 2002
Development Time 3 weeks
Content Length 1 hour
Quality of Outcome ★★★
Description

This game breaks up the *War of the Worlds* radio play (Welles 1983, based in turn on Wells 1898) into 7 mini-games. The events covered: interviewing scientists at the crash , firing artillery, escaping the city, dodging down a highway, crushing a city as an alien, flying as airborne bacteria, and playing as a bacteriophage inside the alien's body to destroy it from the inside.

Key Takeaways

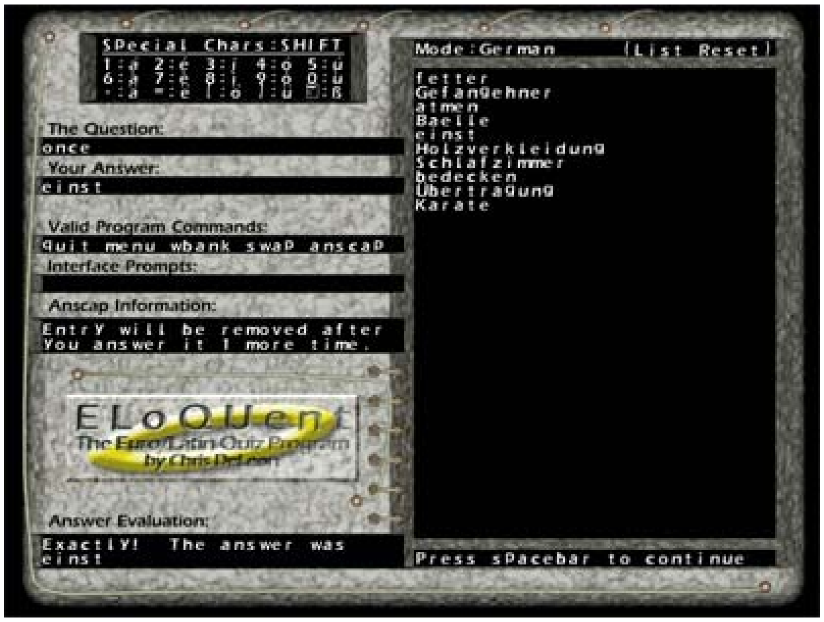
- When designing a game for non-gamers, the game's presentation style and quality of in-game instruction become substantially more important.
- Working to a "license" (esp. non-game IP) motivates some unusual design decisions, but it also provides a wealth of inspiration regarding events, characters, and settings.

Application of Learning

Although I never created another 'mini-games collection' style project, I have applied the minimalist design philosophies learned here to help streamline beginner Game Creation Society projects several years later. "What's the least that has to happen for this game to be fun, fair, and finished?"

Eloquent

Screenshot



```
Special Chars:SHIFT
1:á 2:é 3:í 4:ó 5:ú
6:à 7:è 8:ì 9:ò 0:ù
-:â =:ê [:ô ]:û [illegible]:ß

The Question:
once
Your Answer:
einst

Valid Program Commands:
quit menu wbank swap anscap
Interface Prompts:

Anscap Information:
Entry will be removed after
you answer it 1 more time.

Answer Evaluation:
Exactly!  The answer was
einst

Mode:German          [List Reset]
fetter
Gefangehner
atmen
Baelle
einst
Holzverkleidung
Schlafzimmer
bedecken
Übertragung
Karate

Press spacebar to continue
```

What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date July 2002
Development Time 3 weeks
Content Length 2 hours
Quality of Outcome ★★
Description

Eloquent is more a study tool than a game. At the core, it's a flashcards program with a text-driven interface. The deck can also be flipped over, switching which half of the matches are used for questions. The program can be set to remove a word from the randomly-drawn question set after getting it right some number of times.

Eloquent comes pre-loaded with dictionaries for French, German, Italian, Latin, Portuguese, and Spanish. The user is also able to input a custom dataset into the program to drill on anything text flash cards can be used for.

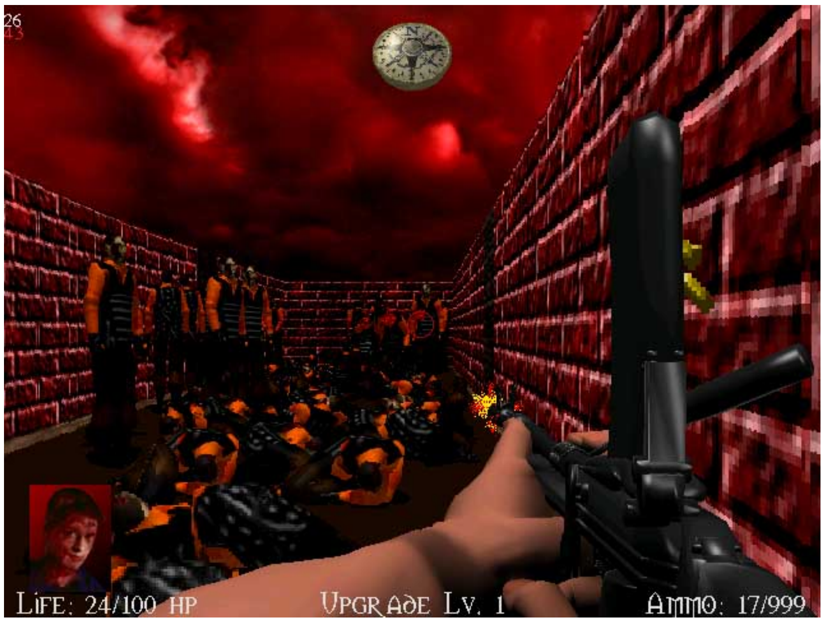
Key Takeaways
- Using text driven menus is insufficient, even for a largely text centered program.
- Pre-rendering a 3D figure for use as the 2D interface makes it dramatically faster/easier to iterate on the design's geometry.
- Technologies and techniques that I learned through videogame development were immediately relevant in developing non-game software.

Application of Learning

I wrote a more complex interface class for **RCC Card Maker** (2005) based on helicopter logic from **The Guinea Pig** (2003). I have used an updated version of that class for every complex interface I've made since.

Guile of Dybbuk (GoD)

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date October 2002
Development Time 7 weeks
Content Length 2 hours
Quality of Outcome ★★★
Description

GoD is a raycast first-person shooter that incorporates mouselook, jumping, crouching, destroyable walls, and a few fully 3D elements.

Weapons included a fire axe, pistol, uzi, heavy machinegun, sniper rifle, flamethrower, grenades, and C4. Monsters included zombies, uzi demons, machinegun robots, and gold robots. Powerups provide ammo, health, and player upgrades. Attacking caused the camera to move based on the recoil/force source. The game's levels were built based on pixel colors in a small image file.

Key Takeaways

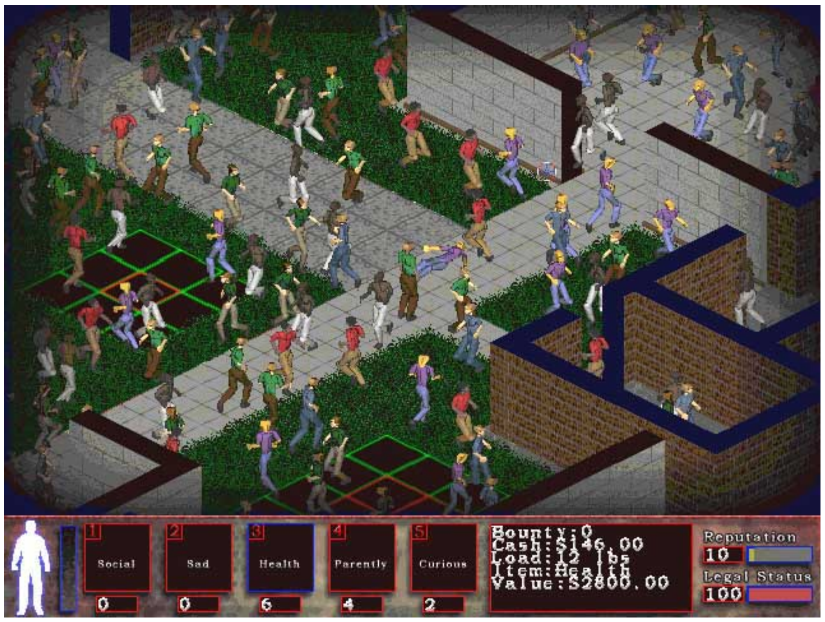
- Putting the player behind the character's eyes makes the experience much less abstract. This leads to fundamentally different priorities during development.
- Sniping frantic zombies and demons in the forest at night with a bolt-action rifle feels memorable, like a twisted dream, despite having no meaningful context.
- Moving camera while attacking added heft, but at expense of gameplay and control. It made the melee more visceral, but felt too gimmicky for the guns.
- If a game lacks an adequate level creation tool, it will lack adequate levels.

Application of Learning

In future projects, starting with **Rowdy Rollerz** (2004), I've made level creation tools more robust. The levels I designed for **Shotgun Debugger** (2005) appealed to the same dream feeling as the nighttime forest sniper setting in **GoD**.

Interactive Android Simulator

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose International Science Fair project
Release Date December 2002
Development Time 4 months
Content Length 1 hour
Quality of Outcome ★★ℸ
Description

IAS received 4th place in Computer Science in the ISEF International Science Fair. It consisted of a core simulation program (loaded pre-saved "people") and a handful of content creation programs. Unbeknownst to my judges, this program included a fully functional machinegun and sniper rifle.

Key Takeaways

- A level creation tool that specializes in room and hallway creation (as opposed to tile placement) makes content creation significantly easier.
- A simple, incomplete simulation can be quite convincing when enough work is put into the presentation layer.
- It doesn't take very complicated A.I. before seemingly intelligent patterns emerge: any NPC to NPC interaction (ex. panicking one another) combined with basic spatial sense (don't run into walls) goes a long way.

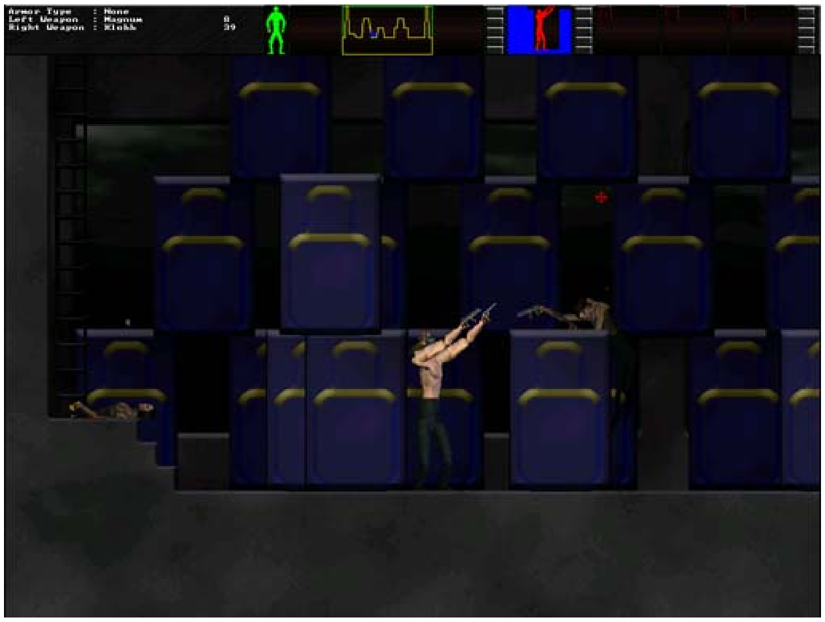
Application of Learning

I never made another per-tile level editor – the units of editing came to be more game design oriented (rooms, walls, roads) and I'd rely upon routines to automatically scan the updated region of the map to update intersection joints or edge pieces contextually. When there is one right solution to a problem that frequently comes up, the computer needs to be made to solve it automatically.

2003

The Guinea Pig

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date March 2003 (abandoned)
Development Time 5 months
Content Length 4 hours
Quality of Outcome ★★★★ɿ
Description

The Guinea Pig is massive, feature packed, and unfinished. Procedural fire, limb-severing damage, destructible levels, weather, parallax, ragdoll, screen rattling, team-commanded AI, flamethrowers, dual wielding , light sabers, upgrades, helicopter flying, medieval melee weapons, 6 towns, 13 continents…

Key Takeaways

- Features in a game are like musical instruments in a band – they need to play nicely together, each filling a complementary niche, and the only way to get away with having a ton of them is to coordinate groupings of well-understood classics. **The Guinea Pig** just had too much going on. More seemed like better.
- I underestimated the difficulty of filling characters with RPG-type dialog without it getting too contrived, annoying, or terse.
- Producer obligation #1: Ensure that the game being planned can be finished.

Application of Learning

Art I made from **The Guinea Pig** made it into numerous other games. Technology from **TGP** its way elsewhere, and lessons from its game design and scope issues have affected every project I have touched since.

One of These Things is Not Like the Other

Screenshot

What I Did Solo project
Other Primary Contributors Tuscan Knox helped Google search for images
Publisher/Purpose Sigma Phi Epsilon booth
Release Date April 2003
Development Time 2 weeks
Content Length 30 minutes
Quality of Outcome ★★
Description

One of These Things was a game for Carnegie Mellon University's Sigma Phi Epsilon Carnival booth. This was my first experience working with custom hardware. Each player received 6 questions from a database, each presenting 4 options, the least similar of which had to be selected in a time limit.

Key Takeaways

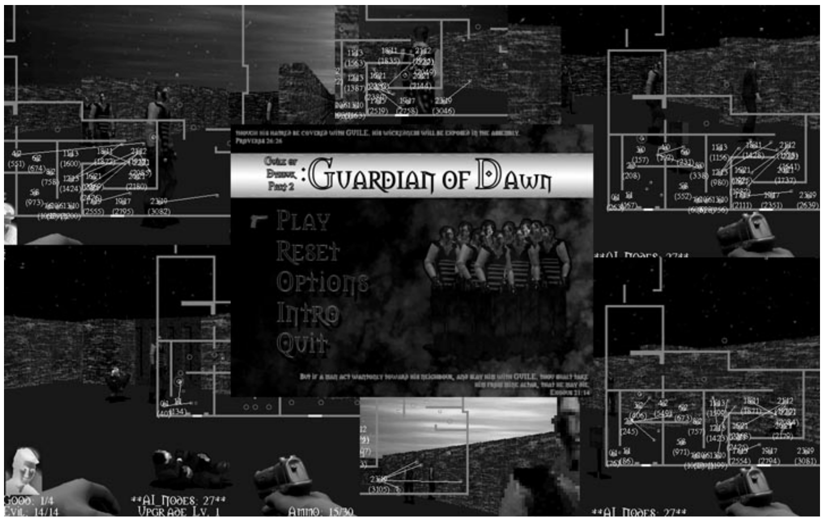
- Providing multiple difficulties (flagging the more difficult questions and reserving them for older players) helped make the game accessible to younger players.
- Tearing apart a joystick and wiring the contacts to other mechanical devices proved far less complicated than I had imagined it would be.
- Little presentation touches, like having the photos float gently and asynchronously from one another in little circles, did a lot to keep the game feeling dynamic, even though it was just pulling questions from a database.
- A game can be put together considerably faster when most players will only see the game one or a couple of times. Replayability is completely a non-issue.

Application of Learning

Future booth games I developed (**Sigma Phi Jamilson** 2005, **Phi Fighter Attack** 2006, **SigEp Trooper Trivia** 2006, and **Super Push 64** 2007) all applied and extended the lessons learned related to making games accessible for all ages and experience.

GoD 2: Guardian of Dawn

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date July 2003
Development Time 4 weeks
Content Length 1 hour
Quality of Outcome ★★★
Description

GoD 1 (2002) was a tech demo and an experiment with setting. **GoD 2**, by comparison, was a gameplay experiment and chance to play with AI pathfinding, built upon the same tech and content as **GoD 1**. Instead of centering on self-preservation, **GoD 2** puts the player in the shoes of an invincible guardian angel, and focuses gameplay on defending helpless civilians from roaming evil hordes, while darting about collecting ammunition for the task.

Key Takeaways

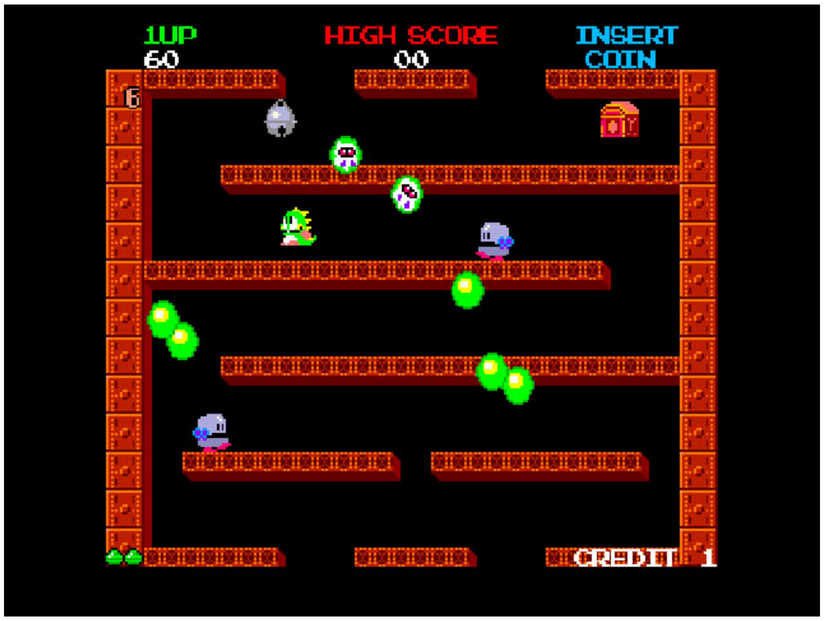
- AI enemies that follow targets beyond line of sight come through as more believable and frightening than those that don't.
- An action game can be quite fun without putting the player in a vulnerable role.
- Embedding asynchronously updated player-visibility information into the path node network worked very well as a batch-processing optimization enabling massive crowds of semi-intelligent enemies to hone in on targets. Effectively, all enemies shared the same brain, creating the illusion of primitive communication.
- As with **GoD 1**: Mapping colored pixels of bitmap data to a level grid is a poorly scaleable level creation solution. Bad level editing tools = bad levels.

Application of Learning

I had a lot of fun with the gameplay model explored here, but have not yet explored that concept in a different game environment. I have, however, re-used the AI solutions from this project in other games, and I have moved away from processing pixel colors in bitmap to generate levels.

Open BubbleBobble

Screenshot



What I Did Solo project, except art/design (used from the arcade game)
Other Primary Contributors Design and art all borrowed lovingly from Taito
Publisher/Purpose Independent
Release Date August 2003
Development Time 1 week (powerups and bosses not implemented)
Content Length 2 Hours
Quality of Outcome ★★★
Description

Bubble Bobble is a Taito game from 1986 designed by Fukio Mitsuji. It involves two dinosaurs navigating one-screen mazes together, spitting bubbles to ensnare enemies, and popping those bubbles before the enemies broke out in anger. I undertook this recreation in an effort to more intimately understand the game's asset requirements, artificial intelligence, mechanics, and level structures.

Key Takeaways

- With modern APIs, programming languages, and hardware, one person should easily do in 6 days what took a team of engineers 6 months or more in the 80's.
- A score number rising out of a collected powerup, combined with a satisfying sound effect, provides a substantially better reward than the points themselves.
- Enemies getting angry when allowed to escape bubbles had a curious psychological effect on the player; enemies that get frustrated seem more real.

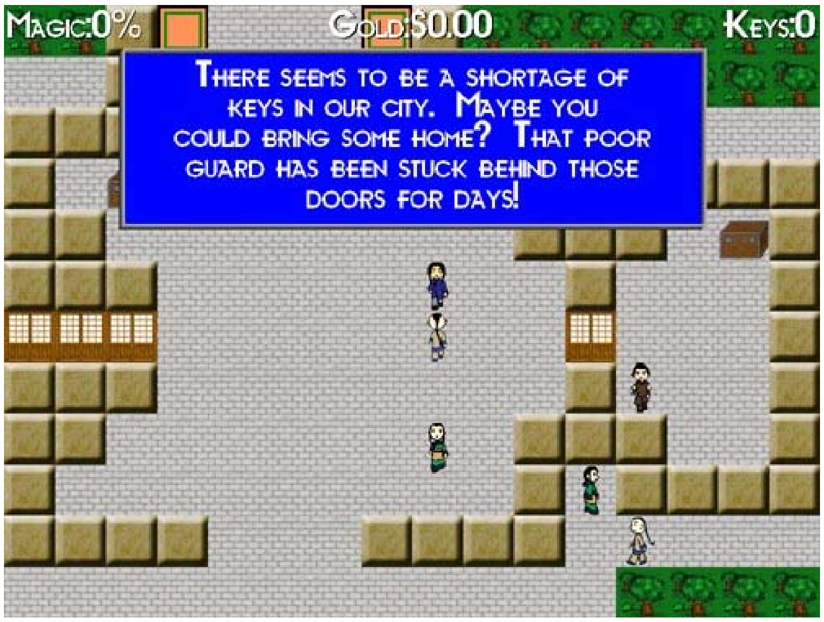
Application of Learning

Each level uses its own grid of orthogonal vectors to dictate bubble drift. That structure led to **Rowdy Rollerz** (2004), in which the player controls a ball by flipping the horizontal or vertical drift grid elements. Variations on parameters affecting its mechanics per-level prompted me to work that versatility into **Hatchling Smasher** (2005).

2004

Dragon of Shiuliang

Screenshot



THERE SEEMS TO BE A SHORTAGE OF KEYS IN OUR CITY. MAYBE YOU COULD BRING SOME HOME? THAT POOR GUARD HAS BEEN STUCK BEHIND THOSE DOORS FOR DAYS!

What I Did Producer, programmer, part of design committee
Other Primary Contributors Tuscan Knox (music), Priscilla Kim (art, writing), Dongha Lee (sound effects), Ryan Tarpine (partial SDL port)
Publisher/Purpose Game Creation Society
Release Date May 2004
Development Time 3 months
Content Length 1 hour
Quality of Outcome ★★⯪
Description

Our only goal was to make a playable and complete RPG spoof. As the first finished Game Creation Society (www.gamecreation.org) project, its functional completion was enough to jumpstart the group to future success.

Dragon of Shiuliang is a lock-and-key RPG with scripted in-engine cinematics, cities with chatty civilians, and two boss battles. Battles were done as Rock-Paper-Scissors.

Key Takeaways

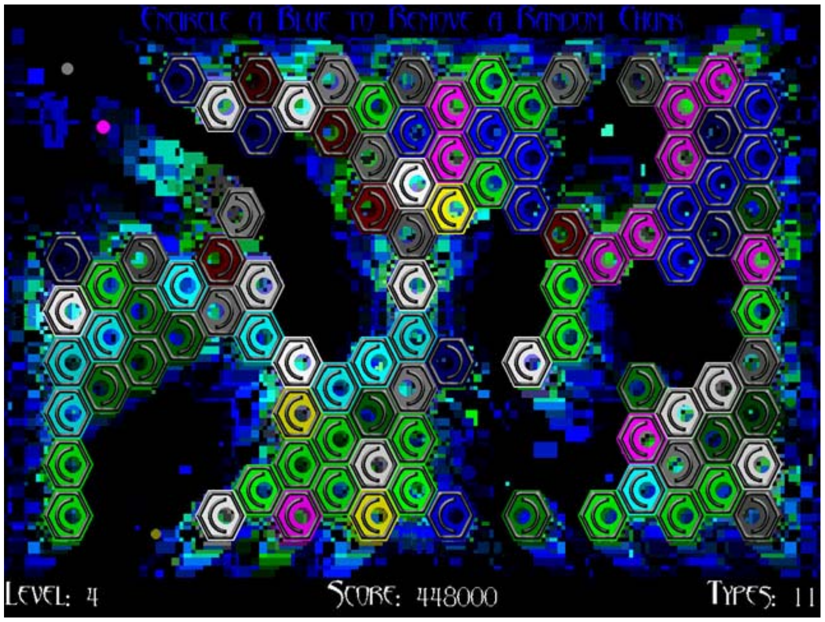
- Writing in-game dialog is difficult for me – something I knew from **The Guinea Pig** (2003), but working on **Dragon** helped drive that point home for me.
- Leaving the level editor hidden in the game as an end-of-game reward was perhaps our best decision across the entire project. It's more fun than the game.
- Artists are even worse than engineers at projecting their own deadlines.

Application of Learning

I've finished 20 projects since **Dragon of Shiuliang** (2004), and not once have I touched in-game dialog again.

SpellCaster

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date July 2004
Development Time 3 days
Content Length 1 hour
Quality of Outcome ★★★
Description

SpellCaster is an original puzzle game concept where the goal is to create hexagonal rings of 6 distinct colors. Which color is at the center of a ring determines which spell gets cast on the entire grid. The player holds one tile at a time, and clicking on a tile causes it to swap out. Exponentially many points are subtracted for each subsequent pointless move, encouraging the player to act efficiently, but the game otherwise presents no time or enemy pressure.

Key Takeaways

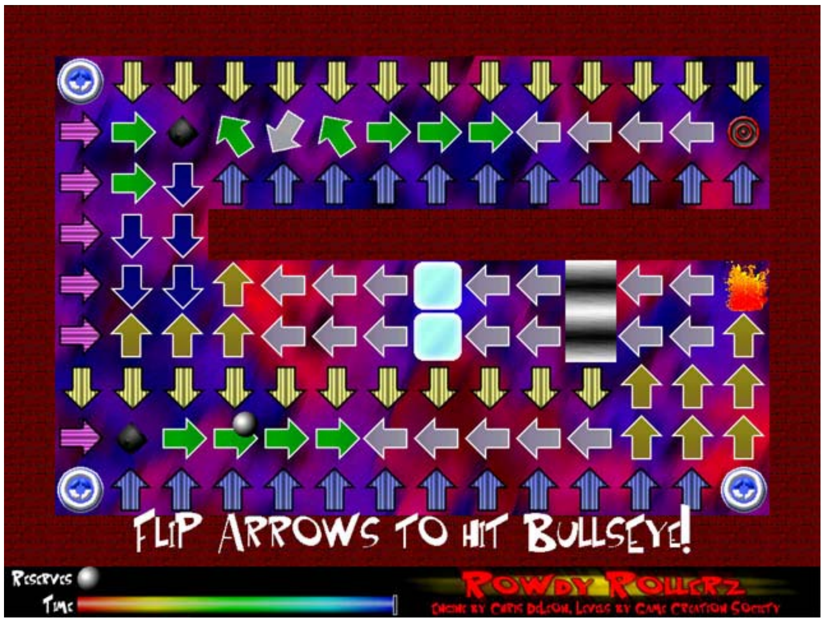
- Developing puzzle games calls for an entirely different type of thinking.
- Generating different tile patterns for each of the game's 10 levels helped reward player progress, and prevented the game from feeling too monotonous.
- I was pleased with the alchemic nature of the gameplay, in which the in-game manual only vaguely alluded to what spell each tile would cast when encircled, forcing the player to learn by trial and error. Complexity of chain reactions became a core emergent gameplay mechanic, making it challenging for even me to decipher which tiles were best to ensnare in certain situations.

Application of Learning

Practice writing the level generation algorithm for this game paid off when I needed to help a GCS team with a similar task for **Swapping Action** (2005).

Rowdy Rollerz

Screenshot


FLIP ARROWS TO HIT BULLSEYE!
RESERVES
TIME
ROWDY ROLLERZ
ENGINE BY CHRIS DELEON, LEVELS BY GAME CREATION SOCIETY

What I Did Producer, Programmer, Designer, SFX
Other Primary Contributors Tuscan Knox composed the game's music; levels were made by Kevin Costello, Tom Cauchois, Casey Miller, and Dan Perkins
Publisher/Purpose Game Creation Society
Release Date October 2004
Development Time 2 months
Content Length 2 hours
Quality of Outcome ★★★ı
Description

Rowdy Rollerz is plain weird. It's a puzzle-action game, in which the player only has three buttons: flip all horizontal arrows, flip all vertical arrows, and fire (when inside rotating cannons). Each level then puts time pressure on the player and a limited number of lives, with which the player has to navigate unmovable arrows, teleporters, pinball bumpers, gutters, ice obstacles, jump pads, brick walls, lava, and spin cannons.

Key Takeaways

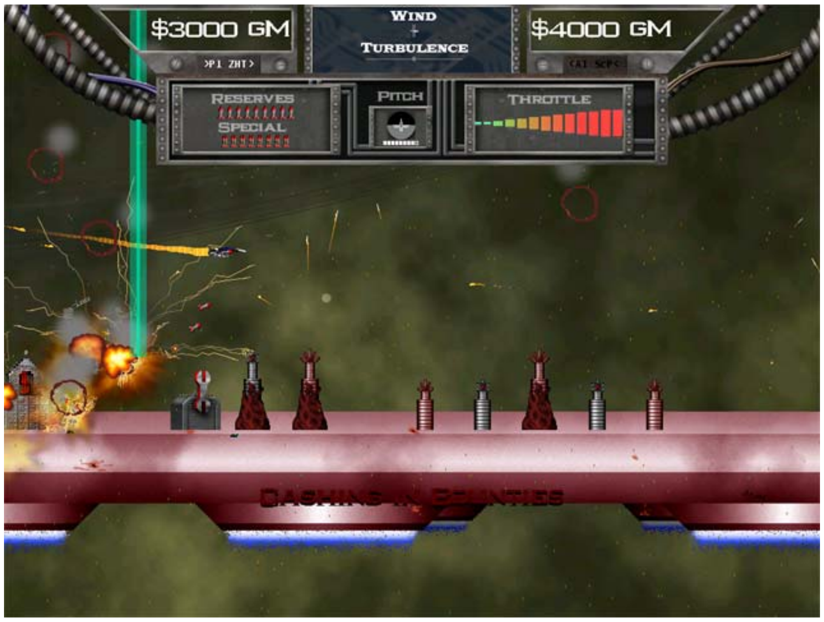
- When a sprite needs to be smoothly animated, writing a tiny program to handle that animation procedurally in preprocessing saves a lot of time.
- The in-game editor allows anyone interested to make levels for the game, which played an important role in getting first time level designers involved
- If a designer creates puzzles balanced for his/her own play, it's probably unfair to the point of frustration to everyone else.
- When levels can be rapidly created, starting fresh is better than iterating on junk.

Application of Learning

Lessons learned about the creating the in-game level editor for **Rowdy Rollerz** (2004) effected the design for level editors used in-game with **BRAD** (2005)**, Eternal Storm** (2006), and **Super Push 64** (2007).

Saturn Storm

Screenshot



What I Did Producer, Programmer, Sprite Artist
Other Primary Contributors George Shannon (Writer, Background Art), Matt Sarnoff (Special Effects Programmer), Alejo Grigera (Interface Artist), music by Tuscan Knox and Chris Brust, with art by John Nesky, Josh Schnarr, and Roy Concepcion
Publisher/Purpose Game Creation Society
Release Date November 2004
Development Time 1 month
Content Length 3 hours
Quality of Outcome ★★★◗
Description

The player flies a ship from one edge to the other in a field 3 screens wide, dropping bombs on defensive buildings below.

Key Takeaways

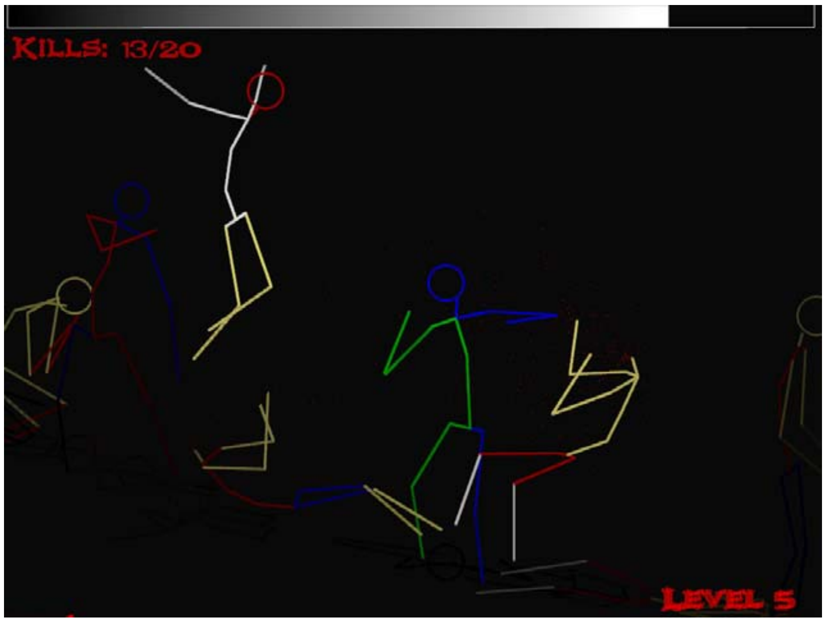
- Desire for bigger explosions, more bombs, flashier strike planes, and increasingly intense enemy defenses all played a major role in keeping players interested.
- The in-game economy for buying ships and bombs added a measure of unqie ownership and personalization to every player's experience.
- Two player mode not only benefited replayability, but also helped our number of players grow by giving today's players an easy way to introduce it to tomorrow's.
- Creating base logic – e.g. having repair buildings relieve damage until destroyed – added a layer of complexity to an otherwise somewhat arbitrary pile of targets.

Application of Learning

The swooping cloud effect from the title screen found its way into **Battleship 88** (2005) as the water, **Sigma Phi Jamsilon** (2005) as notes and bubbles, and **Eternal Storm** (2006) as the clouds. Success with in-game economy motivated a similar feature in the sequel (**Eternal Storm**). Special effects as a compulsion played a major part in **Ghosts in the Machine** (2007).

Mind Breaker v1.21

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Independent
Release Date December 2004
Development Time 1 month
Content Length 2 hours
Quality of Outcome ★★★★
Description

Although collision detection is in 3D (based on intersections between stickman geometry), all movement in **Mind Breaker** takes place in a single left-right plane. The game features six default characters plus one hidden boss. Characters each represented a substantially different fighting style (flying, crawling, boxing, karate, gangster, twisted demon…), and attacks vary upon left/right, front/back, and punch/kick, plus two throws. Blocking happens by performing a matching attack before the opponent's blow landed.

Key Takeaways

- Writing my own 3D animation/interpolation tool was an interesting exercise in 3D math and input. The kind of exercise I'd rather never do again.
- Featuring both co-op and vs. modes for two player action was a wise decision, and didn't take much more effort after single player mode already worked.
- Jittering 3D coordinates each frame created an organic, dynamic style.
- Causing defeated stick figure heads to pop into red particles was a clear way to communicate "This stick person isn't getting back up."

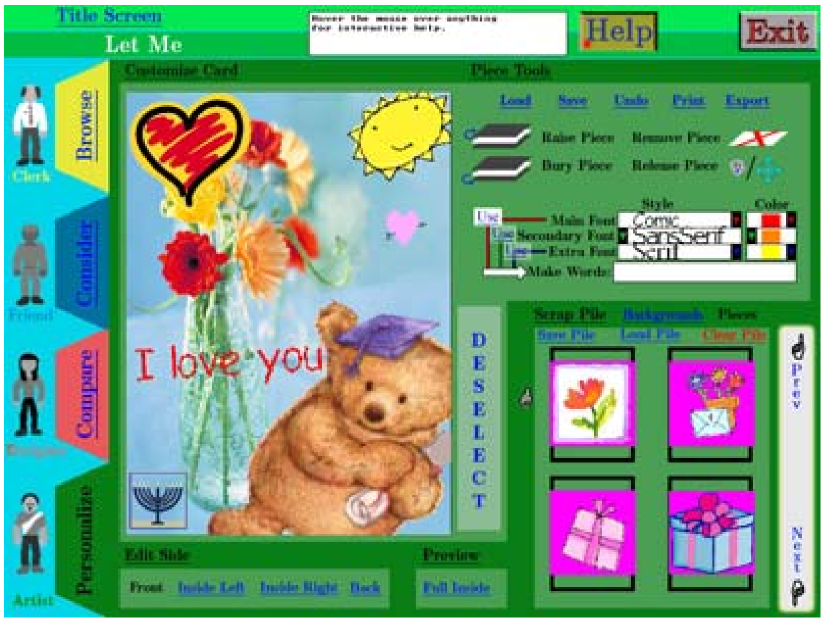
Application of Learning

I borrowed the flicker style for **Ghosts in the Machine** (2007), and used a rotating camera effect from this game for the race end in **Super Push 64** (2007).

2005

RCC Card Maker

Screenshot



What I Did Programming, 1 of 3 designers
Other Primary Contributors Rob Delmont and Constantine Kousoulis (co-designers)
Publisher/Purpose Human-Computer Interaction class project
Release Date April 2005
Development Time 2 months
Content Length 1 hour
Quality of Outcome ★★
Description

RCC Card Maker is a tool to create, preview, and export greeting cards for print. The tool supports imported clip art and rotated/stretched text, as well as project file save and load..

Key Takeaways

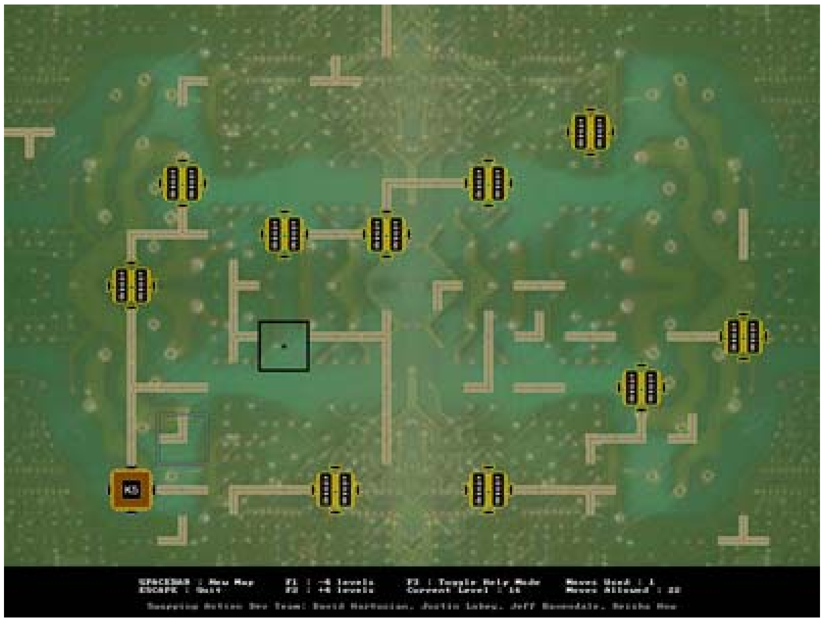
- Interface design for tools can be considerably more daunting than interface design for games. Making options work isn't the hard part – communicating to the user how/where those options are utilized is where the problem gets difficult.
- Complex interfaces are best labeled/sectioned by an in-game tool, instead of trying to program all of those coordinates into an external file by hand based on a screenshot (something I did in projects before this one).

Application of Learning

I reused the tiny interface library that I wrote for this program in **Sigma Phi Jamsilon** (2005), **Battleship 88** (2005), **Hatchling Smasher** (2005), **BRAD** (2005), and **Eternal Storm** (2006). Some of the basic object manipulation input strategies learned in this project affected how I created the level editor for **Super Push 64** (2007).

Swapping Action

Screenshot



What I Did Debugging, Procedural Level Algorithm Assistance
Other Primary Contributors Programming by David Hartunian, with art by Keisha How, Justin Lokey, and Jeff Baxendale
Publisher/Purpose Game Creation Society
Release Date April 2005
Development Time 1 day (with added time for art before my involvement)
Content Length 1 hour
Quality of Outcome ★★
Description

Swapping Action is the brainchild of the GCS members listed above. The game starts with a network connecting all nodes to the source, and scrambles the pathways by randomly switching tiles. The player's goal is to reconnect the entire graph in as few turns as possible by clicking on one tile after another to swap positions.

Key Takeaways

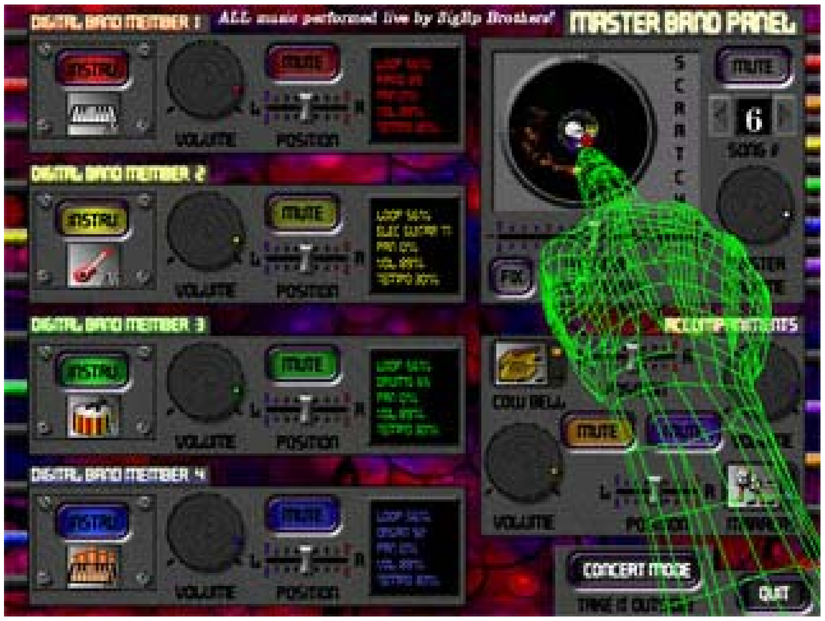
- A simple game, especially one requiring relatively few assets, can be assembled in its entirety in a day, provided a developer has sufficient practice going into it.
- Having a minimalist interface is nice. Generated content is also great, perhaps even more for its affect on the development cycle than what it offers the player.

Application of Learning

Helping Dave Hartunian complete the programming for the project (we met up and talked through it) served as my first experience trying to teach someone else game development concepts and practices. This served as my basic model for instruction when I took on 4 students two years later.

Sigma Phi Jamsilon

Screenshot



What I Did Solo project
Other Primary Contributors None
Publisher/Purpose Sigma Phi Epsilon booth
Release Date April 2005
Development Time 1 week
Content Length 30 minutes
Quality of Outcome ★★½
Description

Sigma Phi Jamsilon is a real-time mixing table, in which the user could select up to four instruments to perform one of 7 core songs. The user can then set up relative volumes and panning, complement the song with cowbell or maracas, and tinker with tempo or "scratch" on the record. The game also presented a "Concert mode" which featured random SigEp brothers performing silly dances on the quad with a few frames of animation each.

Key Takeaways

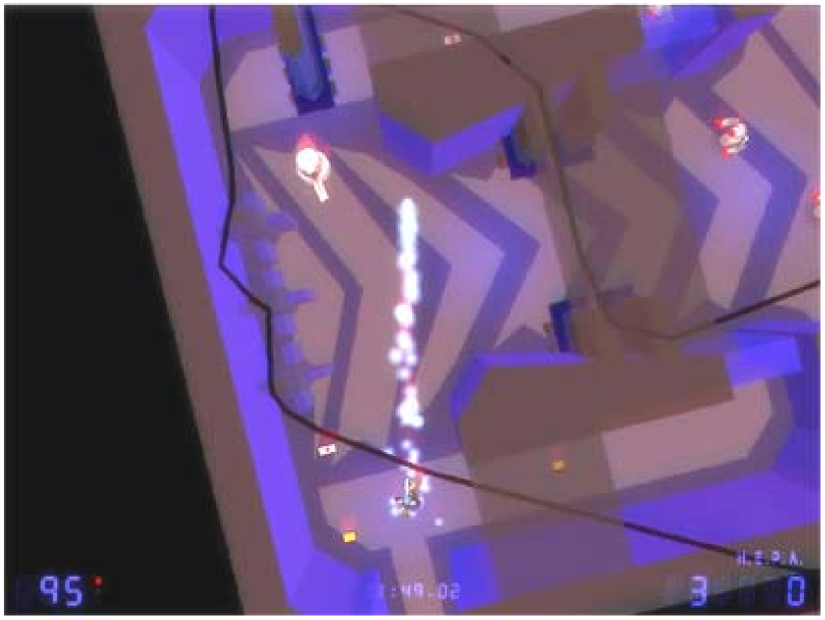
- Technical difficulty or production focus is rarely a good indicator of how well a crowd will respond to a feature. People seemed considerably more fascinated by the pre-rendered wireframe hand cursor, and the ability to "scratch" the music in real-time, than they seemed about the game's core technical achievements.
- The "Concert mode" was rushed in the morning that we installed the game into the booth. As the only person on the project, it was easy to take calculated risks.

Application of Learning

Some of the real-time audio manipulation used in this program found its way into **Battleship 88** (2005), **Ghosts in the Machine** (2007), and **Super Push 64** (2007) to convey intended changes in game intensity.

Shotgun Debugger

Screenshot



What I Did Lead Level Designer, 1 of 2 Game Designers, sound effects
Other Primary Contributors Matt Sarnoff (Lead, Concept, Programmer), John Nesky (3D Model and Textures Artist, Mac OS X port), Tuscan Knox (Music), with Levels by Greg Peng and Jeff Thoene
Publisher/Purpose Game Creation Society
Release Date May 2005
Development Time 4 months
Content Length 3 hours
Quality of Outcome ★★★★
Description

Shotgun debugger began as a 2D game inspired by the classic **Berzerk** (Alan McNeil 1980), and it stayed true to its classical top-down shooter roots while upping the ante in terms of presentation and gameplay depth.

Key Takeaways

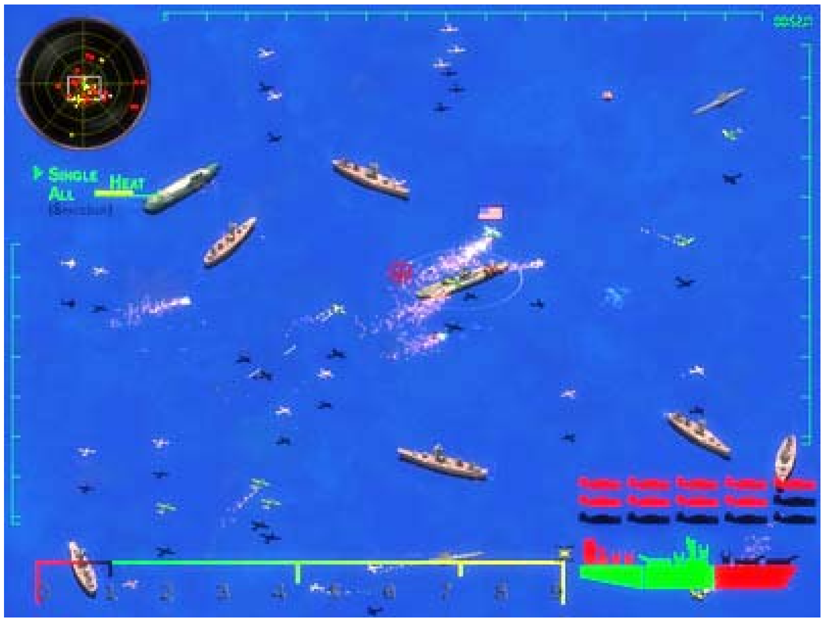
- Level ornamentation is every bit as important as the functional layout.
- Coming up with fictional purposes with each room and level helps steer decoration and color themes.
- Trying to place ammo or health supplies in a level before playing through it with enemies present is largely a waste of time. Complex factors interrelate between AI, environment, and behavior that affect appropriate pacing for such supplies.
- Prototyping and conceptualizing level spaces on paper not only saves time, but it also leads to levels which would not have been conceived in the tool itself.
- When content is cheap to create, overproduce and only keep the best results.

Application of Learning

I carried my learning and level design methodologies from this project into **Eternal Storm** (2006), and **Trichromic** (2007).

Battleship 88: Iron Hero v1.21

Screenshot



What I Did Producer, Programmer, Game Designer, Manual/Documentation
Other Primary Contributors Josh Schnarr (3D Artist, Design), Joe Pfeil (Sound), David Astle (Initial Concept), www.bangbangzoom.net (Music)
Publisher/Purpose Game Creation Society
Release Date May 2005
Development Time 5 months
Content Length 10 hours
Quality of Outcome ★★★★★
Description

25 levels packed to the brim with Kamikazes, Torpedo Bombers, Battleships, Destroyers, Aircraft Carriers, and Submarines. The gameplay expanded on the dissociated movement-targeting model established by **The Guinea Pig** in 2003. With well over 100,000 downloads worldwide, **Battleship 88** is by a wide margin the most popular and successful game I have worked on.

Key Takeaways

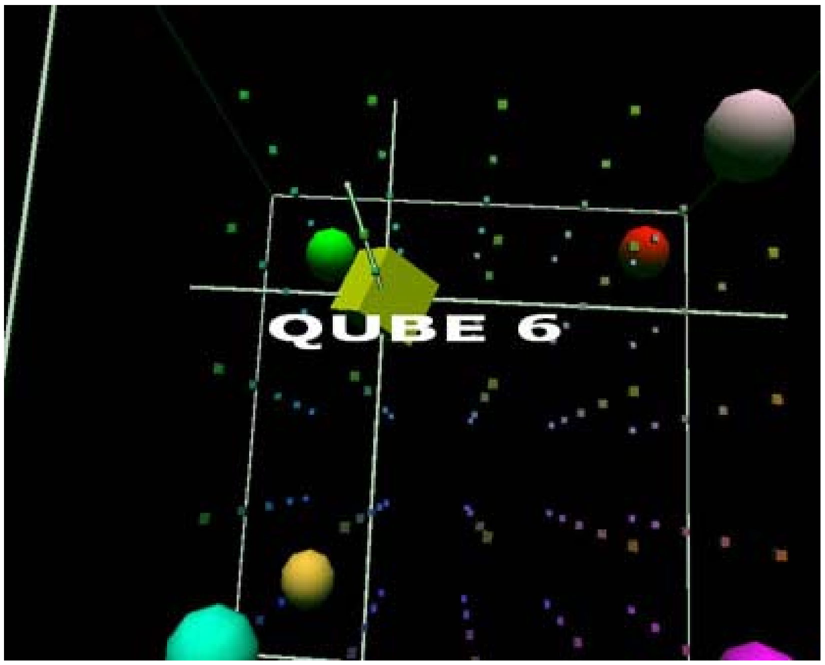
- Playtesting *really* paid off for this game, as it resulted in a handful of easily added updates that made a big difference in the game's first-play comprehensibility.
- Unlockable Cheats helped replayability, and required minimal development time.
- Good music can do a lot for a game's atmosphere. In this case, we worked out a deal with www.bangbangzoom.com at the 2005 Game Developers Conference.
- Even having relatively few, simple powerups can add depth to an action game, since it motivates the player to perform non-combat maneuvers while under fire.
- This design document was one page, front and back. We triaged **B88** from a larger failed project, and on a reduced timeframe. The result? Focus!

Application of Learning

Trichromic (2007) and **Ghosts in the Machine** (2007) licensed music.

Qubestraphobic

Screenshot



What I Did Solo project
Other Primary Contributors Raphael Mun for the **3Dreams** engine
Publisher/Purpose Early demo for **3Dreams** engine
Release Date July 2005
Development Time 1 day
Content Length 30 minutes
Quality of Outcome ★★½
Description

There have been many attempts to adapt **Pac-Man** style gameplay into 3D space. What makes **Qubestraphobic** different is that it attempts to substitute dimensionality for walls, and a dynamic floating camera to maintain a sense of depth across the ever-smaller (by level) 3D matrix of pellets. This game was primarily done as a demonstration of an early form of Raphael Mun's **3Dreams** engine, and it was created with zero special assets (primitives only).

Key Takeaways

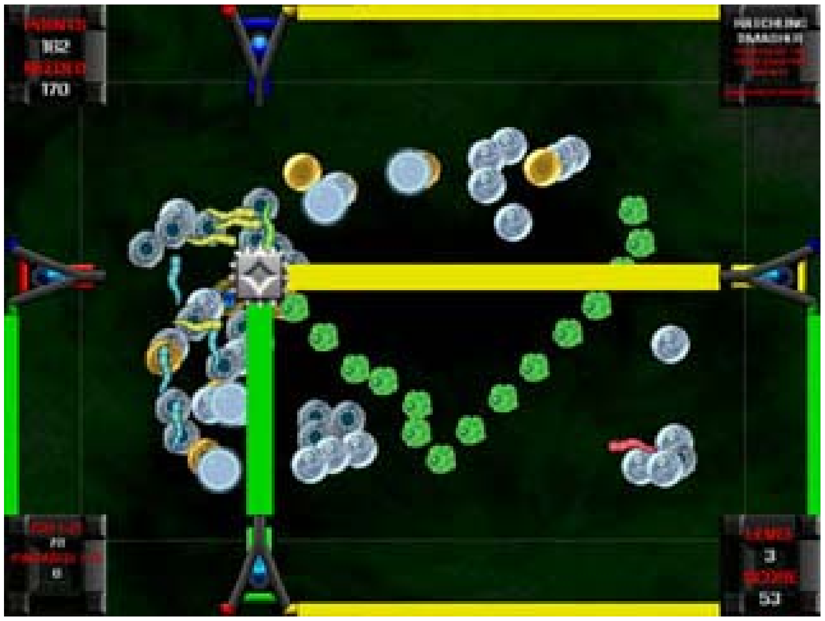
- Controlling a 3D camera turned out to be an order of magnitude simpler than I assumed it would be.
- AI that semi-randomly changes state (chase, wander, wait) at random intervals establishes much deeper dramatic tension, presumably due to the unpredictability, or perhaps the illusion that they're intelligently conspiring against the player.
- A game with a sufficiently light/empty asset pipeline can easily be put together in a day.

Application of Learning

The key tapping input and the look of the axis which identified the player's cursor position inspired the input used for **Hatchling Smasher** (2005).

Hatchling Smasher

Screenshot



What I Did Producer, Programmer, Game Designer
Other Primary Contributors Ross Lafond (Music), Seth Boyles (Art), Robert Strickland (Art, Game Design), level/powerup/boss designs by Greg Newby, Andy McKelvey, Se Yeong Byeon, Michael Hartwell, Mark Levine, and Greg Hallenbeck
Publisher/Purpose Game Creation Society
Release Date October 2005
Development Time 2 months
Content Length 2 hours
Quality of Outcome ★★★
Description

Hatchling Smasher is a game where the player guides a tiny spiked instrument around inside alien-infected organs. Each input arrow causes its opposite-side laser to fire, delivering a strong impulse to the brick. This project experimented with boss and powerup design, working with a team of inexperienced (first time) game designers, and creating a data-driven architecture.

Key Takeaways

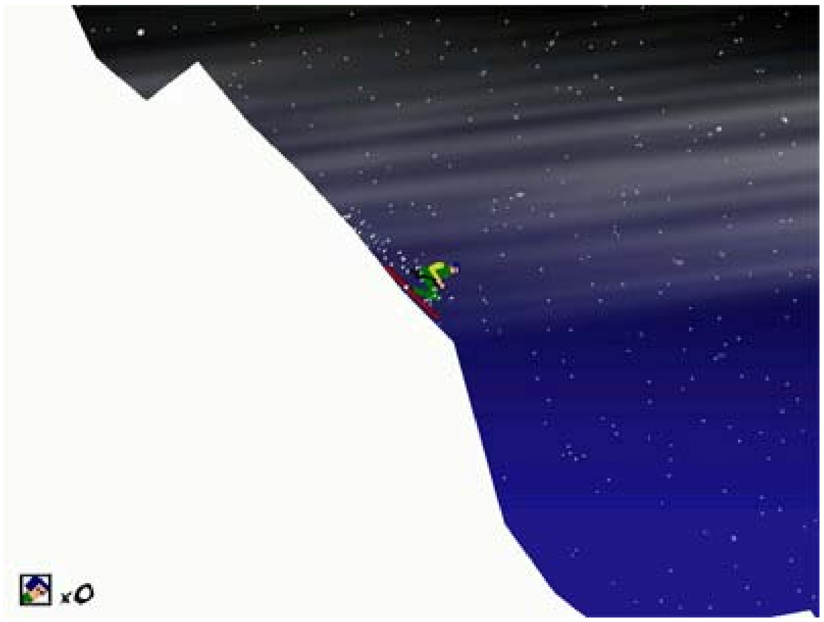
- Two player co-op was an excellent addition to this game.
- I built support for animated sprites into my Excalibur engine (my data driven architecture), and that effort has become a real life saver for later projects.
- Working with first time designers to realize their powerup and boss concepts was a fun way to get people involved, and led to many diverse features being added.
- The game has 4 themes: Nervous, Circulatory, Respiratory, and Digestive. Each theme has distinct powerups, bosses, music, and backgrounds, preventing the game from turning into a simple pile of levels, as **Rowdy Rollerz** (2004) did.

Application of Learning

Level themes carried into **BRAD** (2005), **Rocker Madness** (2006), **Eternal Storm** (2006), and **Trichromic** (2007).

Brad and Roxy's Amazing Downhill

Screenshot



What I Did Programmer, Co-Game Designer
Other Primary Contributors Robert Strickland (Producer, Artist, Co-Game Designer, Level Design), Jonathon Brodsky (Art, Music), Yann Seznec (Music)
Publisher/Purpose Game Creation Society
Release Date December 2005
Development Time 1 month
Content Length 1 hour
Quality of Outcome ★★★
Description

Ski Free (Chris Pirih 1991) has long had a place in the hearts of gamers, but what most people remember about the game isn't doing flips off rocks for points, or winning slalom – what people remember is being chased and eaten by the snow monster. We decided to make a game focused on the chase, and present it in side view, playing more like **ExciteBike** (Shigeru Miyamoto 1984), albeit with plenty of added obstacles to jump over and duck under.

Key Takeaways

- If a level is too difficult to finish, don't let the game get released that way.
- Rubber-banding the chase monster (so it was constantly right behind the player) was a cruel but appropriate way to keep tension high throughout the levels.
- We should have given some more immediate visual or audio indicator for how well the player was matching the slope's angle. No one plays **B.R.A.D.** right.

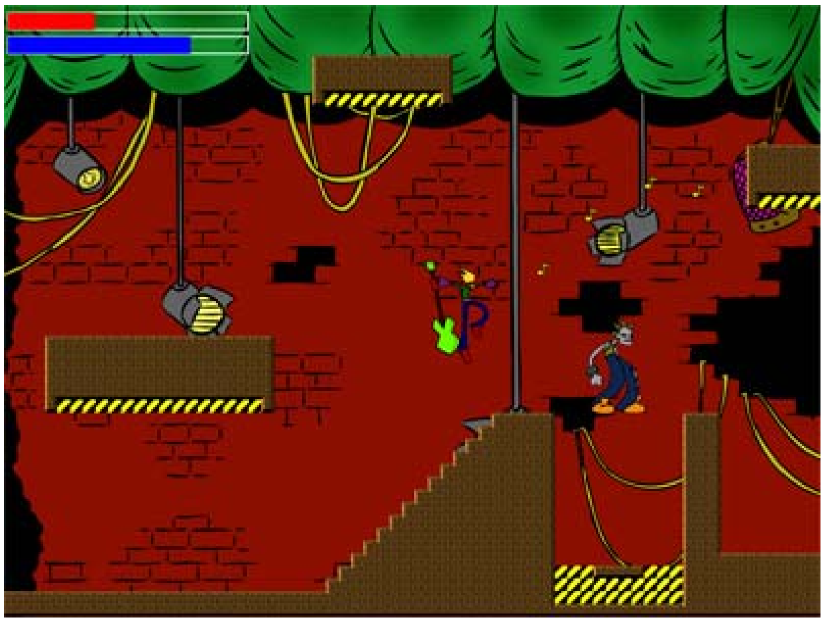
Application of Learning

Rubber banding was present for losing buggies in the original Arcade version of **Super Push 64** (2007) to let people of all ages win, although this was removed for the more challenging PC port.

2006

Rocker Madness

Screenshot



What I Did Development Director, Memory Debugging, AI
Other Primary Contributors Kwasi Mensah (Lead, Programmer), Greg Peng (Lead Level Designer, Sprite Artist), Michael Menchaca (Lead Artist), Eric Barndollar (Audio), Matt Siffert (Music), Levels by Henry Wladkowski, Hamza Taha, Joshua Jelin, Tarun Agarwal, Greg Hallenback, Robert Strickland, and with Art by Theresa Chen, Joe Laquinte, and David Hsu
Publisher/Purpose Game Creation Society
Release Date January 2006
Development Time 1 month
Content Length 1 hour
Quality of Outcome ★★⯪
Description

Rocker Madness features gravity switching, shooting music notes, guitar melee, basic powerups, and a variety of simple enemies (they bounce, float, wander back and forth, etc.). My involvement was fairly limited: guidance early in the project's development, code to automatically bevel/"light" level tiles, and later I helped tie up memory leaks and program for the AI.

Key Takeaways

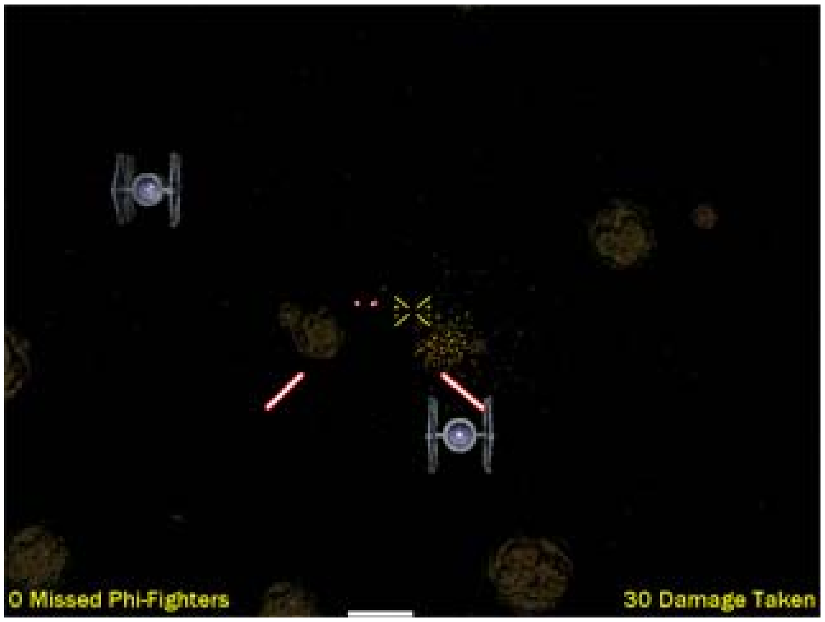
- Allegro is a simple enough library that someone with minimal development experience can put together a complex game with it on their first effort.
- A game which is "98% finished" isn't finished. Do the last 2%, then share it.

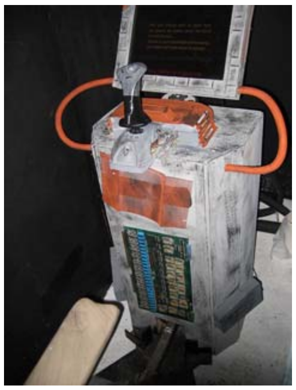
Application of Learning

After this project ran into trouble just short of the finish line, I began a campaign to encourage first-time developers to think small before thinking big.

"Phi Fighter" Attack

Screenshot





What I Did Producer, Designer, Programmer
Other Primary Contributors Bill Lukens (input hardware), Joe Trapasso (voice acting), Dan Ceppos (writing)
Publisher/Purpose Sigma Phi Epsilon booth
Release Date April 2006
Development Time 1 week
Content Length 30 minutes
Quality of Outcome ★★★★
Description

To emulate the Star Wars (Lucas 1977) Falcon scene for this booth game we created a swiveling chair to turn the "Epsilon" Falcon camera, along with alternating top/bottom lasers, allied supporting "Sigma"-wings, swooping "Phi"-fighters, asteroids, and warp-speed effects.

Key Takeaways

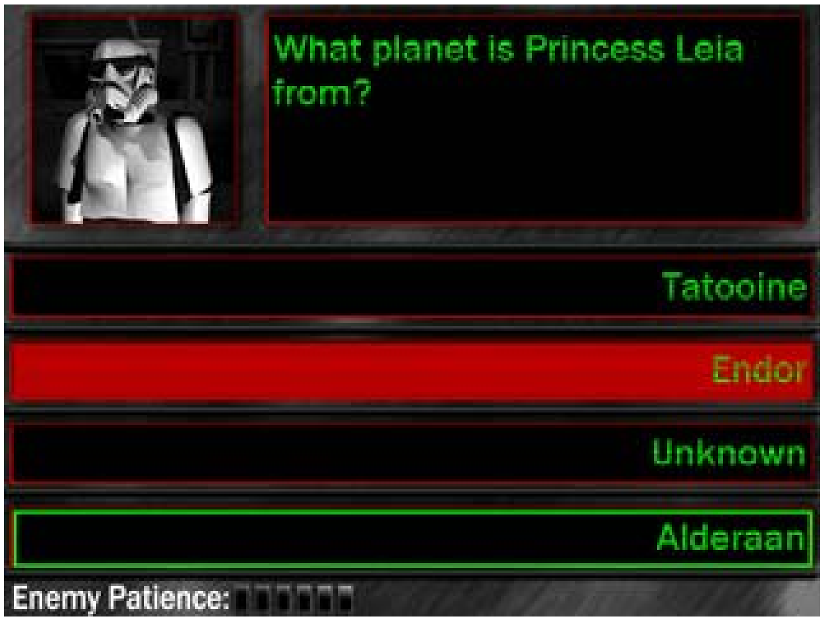
- The input device which allowed the player to pan the screen by swiveling in the stool (screen rotated to face the seat at all times) came across extremely well.
- Never letting the player die/lose turned out to be a good choice for this game. People of every age and ability had to play it. Everyone's a winner.
- Convincing sound effects, voice acting (brothers emulating Chewie growls and Han Solo lines), and music by Holst (*Mars*) helped sell the scene's setting.
- Showing the camera swoop into the turret of the "Epsilon" Falcon when the game opened was a great way to open with a bang, and to establish character.

Application of Learning

Bill Lukens and I teamed up again to put together **Super Push 64** (2007), with him once again providing top-notch and rugged custom input hardware.

SigEp Trooper Trivia v1.0b

Screenshot



What planet is Princess Leia from?

What I Did Programming, 3D Modeling, Animation, Interface Design
Other Primary Contributors Tim Dimond (voice talent), Alan Katz (voice talent)
Publisher/Purpose Sigma Phi Epsilon booth
Release Date April 2006
Development Time 3 days
Content Length 30 minutes
Quality of Outcome ★★★
Description

A simple 6 question trivia game, in which the player is supposedly Han Solo trying to fool a radioing storm trooper into believing nothing is wrong after shooting up the cell block in New Hope (Lucas 1977).

Key Takeaways

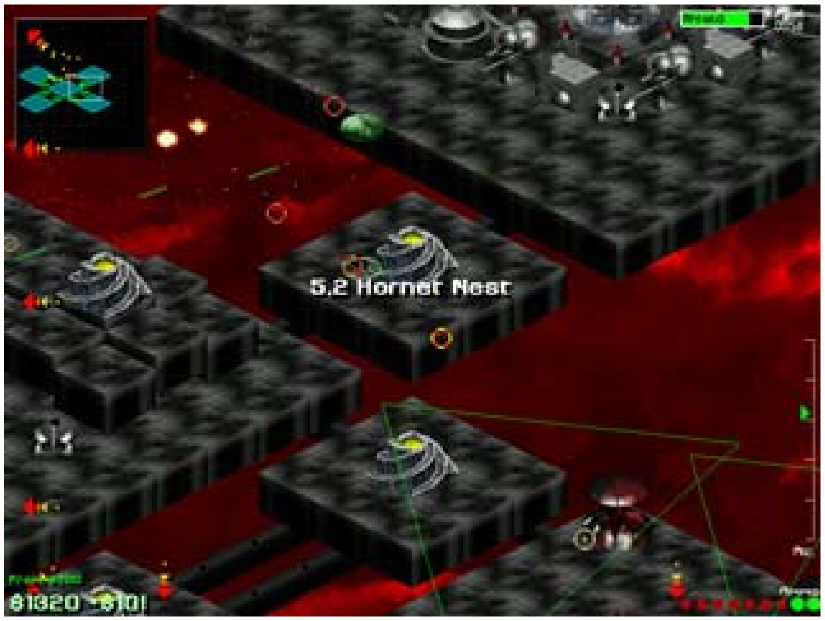
- It's rare that I get to model and animate a humanoid character, let alone as the focal point of a game. It was surprisingly easy and enjoyable, with no flesh/eyes/mouth exposed.
- If you're going to have to animate someone talking, putting their head behind a full mask makes it significantly easier. I animated 3 or 4 generic talking gestures for responses, then played them in randomly rearranging loops while the voice recordings played, and it genuinely looked like the character was talking.
- Nothing does lighting better than pre-rendering frames.
- Flickering black horizontal lines over the green text had a nice terminal feel to it, and that effect was relatively quick and easy to implement.
- The enemy patience meter at the bottom of the screen was a nice way to keep gameplay moving and force the player to focus.
- Using a walkie-talkie between voice talent (Tim Dimond) and the microphone created a perfect storm trooper imitation voice.

Application of Learning

This project boosted my confidence in 3D modeling, which led me to pursue more 3D work in the future, included building renders for **Eternal Storm** (2006), 3D level design for **Power Monkey** (2006), cel-shaded army units for **Trichromic** (2007), and the southern quarter of my college campus for **Super Push 64** (2007).

Eternal Storm v1.2

Screenshot



What I Did Programming, 1 of 2 Game/Level Designers, Cutscene Artist, Voice Acting
Other Primary Contributors Robert Strickland (Level Design, 3D Artist, Voice Acting), Josh Schnarr (Lead 3D Artist, Voice Acting), George Shannon (Writing), John Nesky (3D Artist), Ning Sung Lee (Cursor Artist), Eric Barndollar (Music), Greg Newby (Lead Playtester, Voice Acting), Joe Pfeil (Audio Lead), Matt Kent (Music, Sound Design), with additional sound effects by Joseph Chakola, Joe Kincaid, Megan Sasser, and Danny Traylor
Publisher/Purpose Game Creation Society
Release Date May 2006
Development Time 5 months
Content Length 4 hours
Quality of Outcome ★★★★
Description

Eternal Storm was meant as an extension of the **Saturn Storm** universe, and built on top of the **Battleship 88** engine. It includes voiced-over story sequences between chapters, for five chapters, each combining 3-5 isometric bombing missions with articulated levels and a texture set unique to that chapter.

Key Takeaways

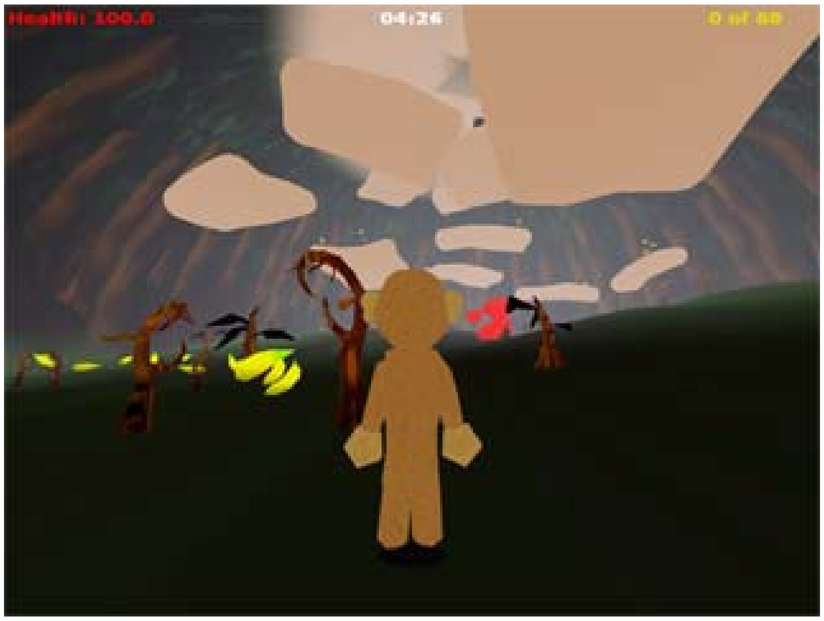
- When experimenting with an original idea, it should never tie up lots of resources.
- Isometric camera was wrong here - players had to fly slowly to see the horizon.
- The ability to buy player upgrades provided a compelling reason to replay finished areas, and also led to more side objectives being explored.
- Rendering at a consistently low frame-rate can seriously affect playability.
- Large team size resulted in time lost to bottlenecks, conflicts, and documentation.

Application of Learning

I've shifted back to smaller, surgical teams for my independent work.

Power Monkey

Screenshot



What I Did Level Design, Sound Effects

Other Primary Contributors Raphael Mun (Producer, Game Design, Programmer), Dan Cuellar (Music), Michelle Hales (Asst. Game Designer), Robert Strickland (Sound Programmer), Jin Kwon (Menu Animations) with Voice Acting and 3D Art by Keisha How, Justin Lokey, and Jeff Baxendale

Publisher/Purpose Game Creation Society

Release Date May 2006

Development Time 7 months

Content Length 1 hour

Quality of Outcome ★★★

Description

Power Monkey, a 3D collection platformer, is mainly a tech demo for Raphael Mun's **3Dreams** engine, and servers as a moving target to lead **3Dreams** to support new features. Newer **Power Monkey** games have been released since.

Key Takeaways

- When shallow AI can't carry a game, filling the levels with collectable powerups and setting strict time limits can make for a pretty decent platformer. Examples: **Super Mario Bros. (**Shigeru Miyamoto 1985**)**, **Sonic** (Hirokazu Yasuhara 1991).
- Under time pressure, the loss of precious seconds is sufficient punishment for missing a jump, without need to otherwise harm or punish the player.
- Providing a range of target times (Gold, Silver, and Bronze) helped replayability by first allowing players to continue on to the final level before fully mastering the early stages, and then giving a clear goal to strive for upon returning.

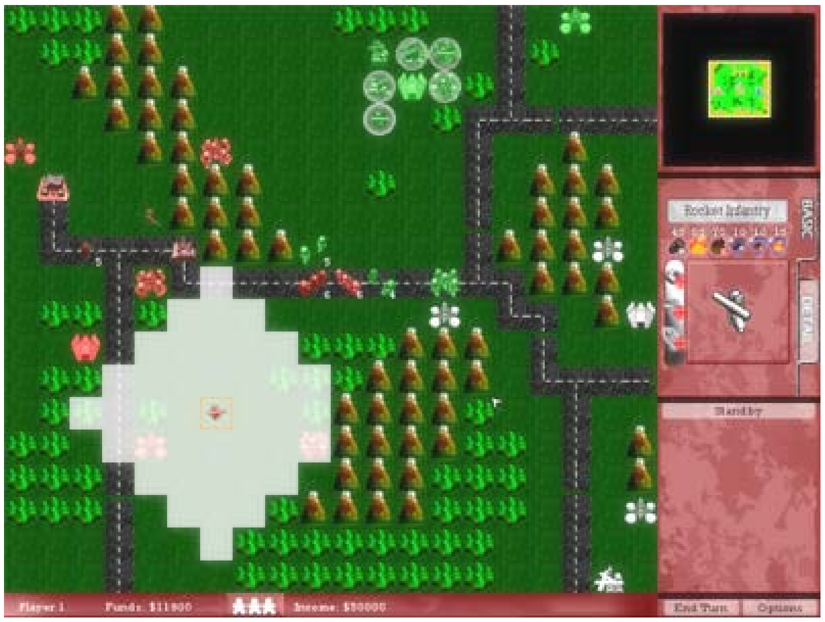
Application of Learning

The gradation of medals for level times motivated the deliberate staggering of opponent buggy pace times for **Super Push 64** (2007).

2007

Trichromic

Screenshot



What I Did Game Design, Unit Design, Level Design, 3D cel-shaded modeling
Other Primary Contributors Kent deVillafranca (Lead/Programmer), Greg Peng (Interface), Sam Chien (Title Music)
Publisher/Purpose Game Creation Society
Release Date February 2007
Development Time 6 months
Content Length 3 hours
Quality of Outcome ★★★★★
Description

The mechanics of Trichromic were directly inspired by **Advance Wars**, but the units, armies, and major gameplay elements are entirely original.

Key Takeaways

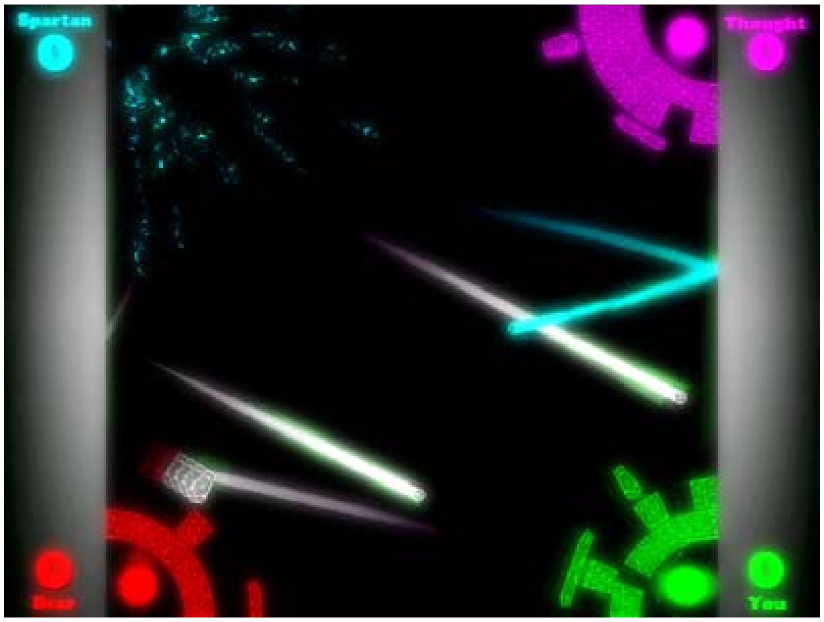
- Asymmetric balance isn't difficult, and it results in considerably more complex multiplayer matches. The trick to getting away with functionally different units under each army was to start with that, then pivot and counter-balance a team's overall unit utility by adjusting army-wide stats like price, armor, and firepower.
- We focused on context sensitive actions, instead of menus, to keep gameplay as brisk as possible. I think this paid off by providing greater accessibility.

Application of Learning

The lessons learned about conveying world-state and playing nicely with player impatience impacted the spirit of **Ghosts in the Machine** (2007). Once example of this is in the option to "calculate winner" option in **Ghosts** when the player has been defeated and the game has been reduced to AI vs. AI. (Trichromic doesn't have this exact feature, but does include a variety of Surrender options).

Ghosts in the Machine

Screenshot



What I Did Solo project
Other Primary Contributors Atari for design inspiration
Publisher/Purpose Independent
Release Date March 2007
Development Time 3 weeks
Content Length 2 hours
Quality of Outcome ★★★★◗
Description

Ghosts in the Machine seeks to accentuate the gameplay thrills established by **Warlords** (Atari 1980). It has an incredibly steep learning curve, due mainly to its deliberately obtuse input mechanics. The mouse controls two paddles simultaneously – moving the mouse forward and backward adjusts the shield (which can lock-and-expand in place), whereas moving the mouse left to right adjusts the gun (which can catch/hold one ball at a time to re-launch). Some players are drawn to the challenge of mastering the input and are instantly hooked; others find the controls too awkward and back off immediately.

Key Takeaways

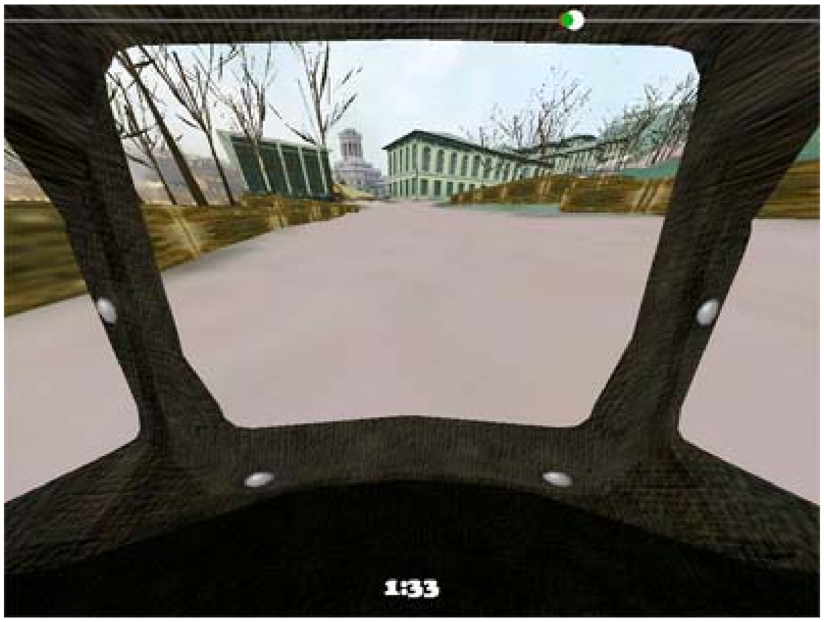
- Using hardware acceleration (via OpenGL) is easy, and makes a great deal of otherwise difficult effects (blur, sharp rotations, silhouette morphing) trivial.
- Many people don't like **Ghosts**, but people that do like it *a lot*. I'm ok with that.
- AI seems most fair when its heuristics steer it to use "human" strategies.

Application of Learning

A surprising amount of engine code from **Ghosts in the Machine** was modified to form the base of **Super Push 64** (2007), including the track data structure (based on **Ghosts** ball trails), optimized polygonal-to-point detection, forced reorientation, and various content/rendering pipelines.

Super Push 64

Screenshot



What I Did Solo project
Other Primary Contributors Bill Lukens (Custom Hardware Input), Vishesh Nandedkar (Raw 3D Data for Buggy Chassis, Paint Job Concepts)
Publisher/Purpose Independent
Release Date April 2007
Development Time 2 weeks
Content Length 30 minutes
Quality of Outcome ★★★★˺
Description

I originally created **Super Push 64** for Sigma Phi Epsilon's Carnival booth, where it used three custom input devices built by Bill Lukens: a push bar, a running mat, and a steering handle. Uphill the player alternated pushing the buggy and running to catch up; downhill the player piloted with the handle. The game features 13 buildings from CMU and the area, along with more than a dozen detail models (statues, bridge rails/fence, barricades…) to bring the course to life.

Key Takeaways

- Occlusion is important to frame rate, even in a relatively small 3D world.
- In terms of wish fulfillment, at least a few buggy aficionados couldn't get over how much it pleased them to see the course from the driver's point of view.
- On-site reference gathering proved invaluable for textures, models, and scaling.
- People enjoy identifying with and relating to real spaces rendered in videogames.
- The "relativistic" camera field-of-view stretch is an idea I borrowed from **Need for Speed: Underground** (Electronic Arts 2003). It conveys speed quite nicely.

Application of Learning

Since this was the last independent game that I finished before preparing this text, I have not *yet* had a chance to apply my learning from it elsewhere...

GAME OVER

(To be Continued...)