Editors: Gitta Domik and Scott Owen

# Practical Game Design and Development Pedagogy

**Paul J. Diefenbach**
*Drexel University*

Unlike many game development programs, Drexel University's program doesn't reside in one department and so mirrors the true nature of commercial game development.[1] Drexel's Digital Media program offers a game art and production major that instructs students on the fundamental skills of design, art, programming, 3D modeling, animation, audio, and video production and on how to use industry tools such as Maya. The Computer Science department offers a game programming and development concentration focusing on software development skills and offers software courses for prototyping game concepts. The gaming courses (www.games.drexel.edu/classes.html) and RePlay Lab (www.replay.drexel.edu) bring these two majors together, with additional participation of students and faculty from other majors including music industry, screenwriting and playwriting, engineering, and business.

The prerequisite courses provide the technical foundation for Game Development Workshops I and II. Workshop I serves as a design and preproduction phase in which teams of four to six students create a game concept and write a one-page "sell" document, an executive summary, a game design document, and a game prototype in 11 weeks. Instructors select the best, most viable games for full production in Workshop II the following term, and teams are merged and expanded. The sequence mirrors the industry in structure and development, using project- and asset-management software, agile development with weekly Scrum sprints, meeting minutes and communication logs, and a milestone payment mechanism in virtual dollars that translates into student grades, emphasizing soft skills.[2]

In presenting the workshops, we've made several recurring observations about students' tendencies that have influenced the workshops' structure and focus. Here, we detail our observations and how we've modified the workshops in the areas of game design and development.

## Game Design

We noticed that students often have creative ideas but miss factors that might at first seem intuitive. So, Workshop I now addresses vague game concepts and pitches. It reintroduces strategy, gameplay, and story to focus on designing games from the player's perspective so that students can assess how this affects code and assets.

### *Fun and Strategy*

Students' sell presentations often discuss story, characters, controls, and so on but don't mention the word fun. When asked how their game is fun, they typically respond with a list of game features, an intricate story, or exciting visuals. Yet features and story alone don't make for a compelling game, and students often don't examine how their ideas translate to the player's perspective. Students must learn to recognize that games must support players with different playing styles, strategies, and reasons for playing.

A recent castle-defense game illustrates providing strategic support (see Figure 1). In this game, the player must set up catapults and other defenses during a layout stage, and then fight attacking enemies while the defenses automatically fire. Although this concept appealed to fighter-style players, the automatic weapon firing permitted the player to assume only a fighter role during the action gameplay.

Two simple changes expanded the player profile to support more strategic fighters as opposed to "button mashers." The first let players add defenses during attacks, but at a cost to their health because they're defenseless while making changes. The second change let players make defenses that must be triggered by the player's proximity, which

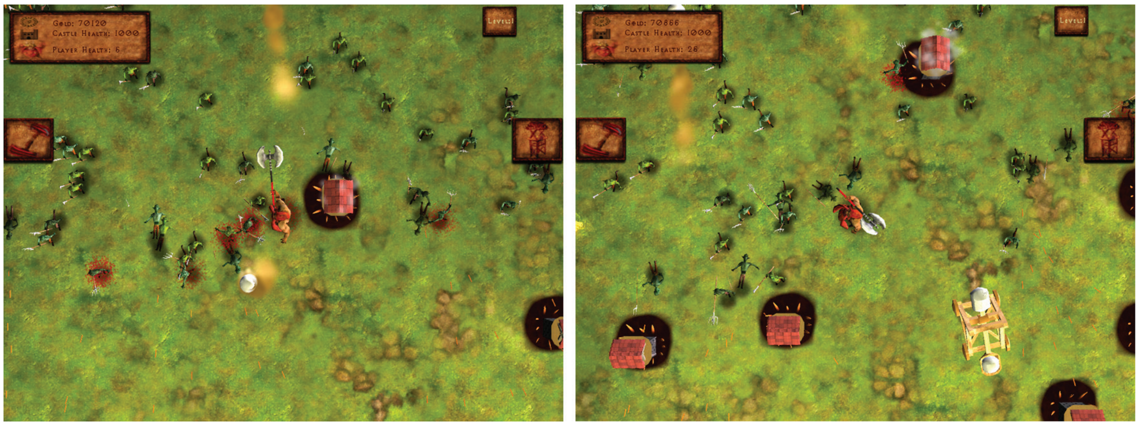
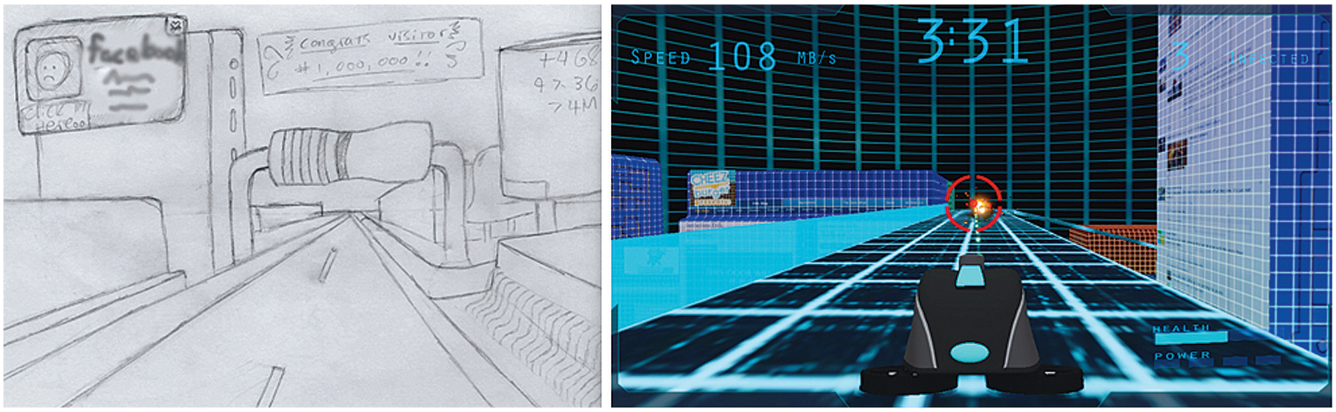

(a) (b)

Figure 1. EpicDefense. (a) The hack-and-slash strategy. (b) The triggered-defense strategy. Workshop students modified gameplay to support different player styles by permitting hack-and-slash play or strategic armament placement and management.



(a) (b)

Figure 2. ZapJak. (a) An early concept sketch. (b) The player's view of the game. The original pitch for this game failed to consider the player's perspective. The students revised their pitch to remedy this problem.

forces players to chose between fighting or triggering defenses.

### *Gameplay*

Students often design games from an abstract perspective, as if the characters in the game are self-governed and the designer is an all-knowing god. For example, they might propose a game about two aliens battling over a magic crystal. The students will have decided the game's look and physical characteristics, the aliens' abilities, the conflict's backstory, and bizarre power-ups, yet disagree when we ask them whether the game is played from a first- or third-person perspective or whether it's played with a keyboard or controller.

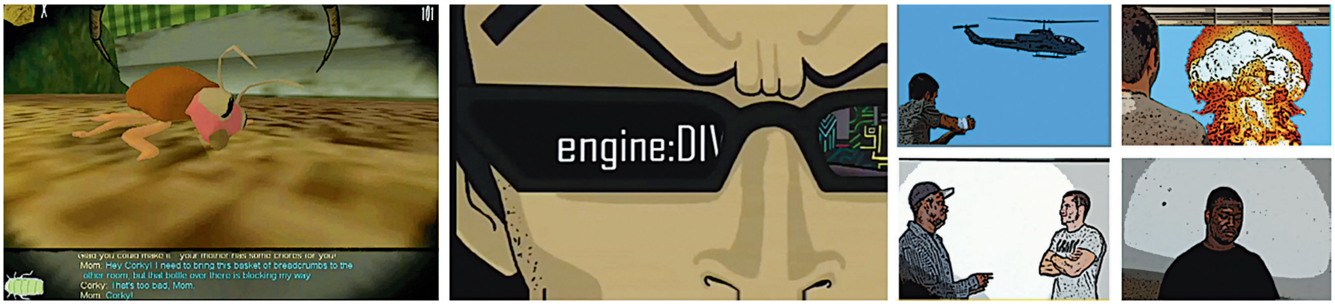
We teach teams to discuss gameplay from the player's perspective. What does the player see? What does the player do? How is information conveyed?

A recent game pitch for ZapJak (see Figure 2) illustrates a typical game description:

> Players race futuristic cars around an Internet-inspired electronic world similar to *Tron* and can destroy or hijack AI bot vehicles.

This doesn't address what the player sees and does. A refined concept that addressed the player's perspective added this:

> The player, using an analog controller stick for speed and direction, controls from a behind-the-vehicle view a futuristic hovercar propelled in a three-tiered city environment of approximately 100 square city blocks. Similar to *Tron*, the game's city roads represent the Internet, and each building represents a website, with larger websites such as Google having larger buildings in the highest, exclusive gated tier. Players use the second analog stick to aim a reticle at enemy bot vehicles to hijack a vehicle and transfer driving control to the new vehicle. With three vehicle types and varied roads



Glad you could make it - your mother has some chores for you!
Mom: Hey Corky! I need to bring this basket of breadcrumbs to the other room, but that bottle over there is blocking my way
Corky: That's too bad, Mom.
Mom: Corky!

**Figure 3. Three approaches to conveying a story. (a) Moach Rotel combined inline dialogue and machinima segments. (b) Engine Divine used cutscenes with hand-drawn artwork. (c) City'Scape also used cutscenes but used Photoshop filters to translate staged photos into a comic-book-style sequence. (Machinama employs the game engine to create an animation; cutscenes are video or animation clips between parts of a game.)**

having different characteristics, hijacking vehicles is integral to player strategy in both timed races and last-player-standing modes.

### *Conveying the Story and Backstory*

Students often also fail to consider the player's inherent knowledge, or lack thereof, of the game. As I previously mentioned, students usually create an intricately detailed story yet fail to consider how to convey the story to the player. Students must be asked, "Does the player see this story as images, video, or machinama? [Machinama employs the game engine to create an animation.] Or must they read text? Or must the game have a voice-over or character dialogue?" Additionally, conveying the story can be resource-intensive for both digital-media artists and computer science software developers, so students must learn to examine the alternatives that work for their game (see Figure 3).

Moach Rotel, a game about a cockroach whose family was killed by the house's owner, was story-intensive, with a 60-plus-page script, in-game dialogue, and machinima segments. Because the characters were cockroaches, the game used clicking sounds for cockroach-speak (obviating the need for voice talent). It presented the dialogue as scrolling text, which was conveyed by a custom dialogue-parsing logic the software developers wrote to translate the scriptwriter's annotated dialogue into in-game events, text, and sounds.

Engine Divine and City'Scape used an alternative strategy: conveying the story by cutscenes (video or animation clips between parts of a game). Both used animatics (an animated storyboard), but each generated the images differently because of their team skill sets and resource allocations. Engine Divine had a strong lead artist and decided that hand-drawn artwork would best convey the complex backstory. The City'Scape team had no strong artist and wanted to convey a comic-book feel for its superhero game. So, the team used Photoshop filters to translate staged photos into a comics-style sequence.

## Game Development

Once the students have vetted their game concept on the basis of the concerns we just detailed, they follow an iterative Scrum development cycle that permits continual refinement. This has introduced problems due to students having little experience in incremental development, milestone planning, and resource management. We now structure the milestones to guide inexperienced students in long-term project planning. Development now involves three distinct phases: establishing the asset pipeline, refining and balancing gameplay, and dressing up the game.

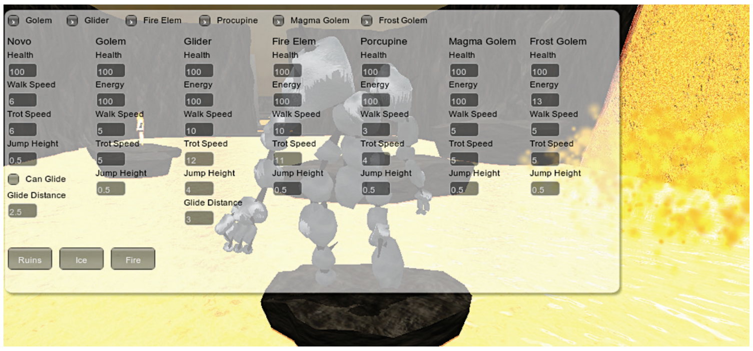
### *Prove the Asset Pipeline Early*

Iterative development has become increasingly popular for games, and Drexel has been teaching Scrum methodology since first offering its game development program. In previous years, teams ran into difficulties because of development issues such as choosing a game engine on the basis of its capabilities instead of the team's comfort with it. Another issue has been the tendency to focus on final models and animations before establishing firm asset pipeline practices or understanding engine requirements, such as scale, polygon and performance limitations, rigging requirements, and bounding geometries.

To minimize potential late-term problems, teams now must make incremental advancements addressing these issues, starting early in the term. Students must produce an in-engine, core gameplay demo by week three and in-game models and animations by weeks five and six, respectively. We don't permit any mods (modifications of a game that require the original release) using game template assets because this can hide asset pipeline issues, such as

- how the bounding geometry's definition can influence collision detection or
- whether animations are exported as bones (which connect joints and control how vertex mesh "skins" deform) or baked into vertex animations.

Workshop II employs a similar approach. Students must demonstrate early proof of competency in creating more complex layered and blended animations and using the motion-capture studio.



Novo
Health 100
Walk Speed 6
Trot Speed 6
Jump Height 0.5
Can Glide
Glide Distance 2.5

Golem
Health 100
Energy 100
Walk Speed 5
Trot Speed 5
Jump Height 0.5

Glider
Health 100
Energy 100
Walk Speed 10
Trot Speed 12
Jump Height 4
Glide Distance 3

Fire Elem
Health 100
Energy 100
Walk Speed 10
Trot Speed 11
Jump Height 0.5

Porcupine
Health 100
Energy 100
Walk Speed 3
Trot Speed 4
Jump Height 0.5

Magma Golem
Health 100
Energy 100
Walk Speed 5
Trot Speed 5
Jump Height 0.5

Frost Golem
Health 100
Energy 13
Walk Speed 5
Trot Speed 5
Jump Height 0.5

**Figure 4. The Proteus game's parameter screen lets play testers tweak gameplay.**

### *Gameplay Balance*

Proving the asset pipeline early doesn't guarantee a successful game; students often don't realize that significant gameplay testing is required. They must test and tweak gameplay factors such as character speed, gravity, the number of shots, and health parameters. However, this tweaking could present a potential bottleneck for the computer science students who adjust these parameters. Because of gameplay balancing's importance, by week five the students must expose all relevant gameplay parameters in the form of an option screen (see Figure 4) that players can modify.

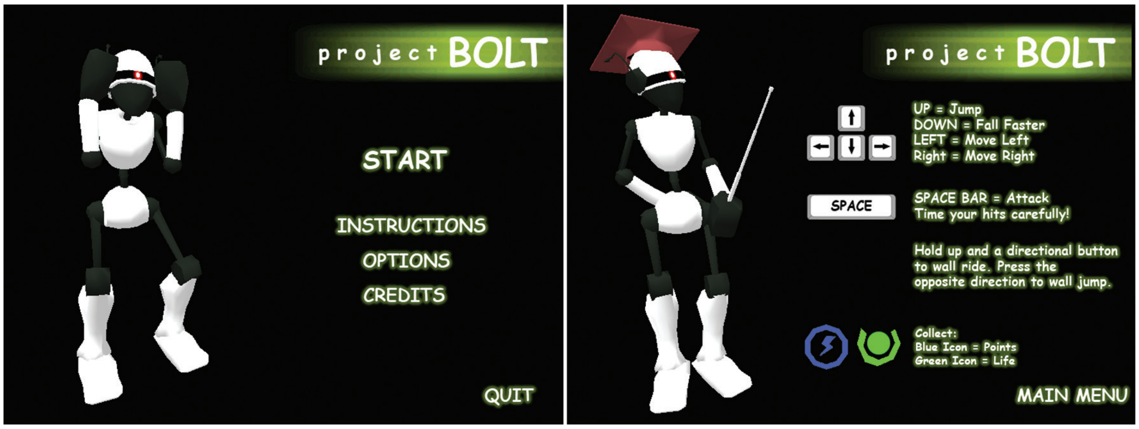
Also, a spreadsheet keeping track of the parameter settings and corresponding gameplay ratings lets developers track optimal settings. This permits every team member to be a tester and facilitates beta testing. An additional benefit is that this player experimentation sometimes results in unexpected play modes, such as when testing found that lowering gravity in City'Scape resulted in a fun exploratory mode, jumping from rooftop to rooftop.

### *The 80/20 Rule*

We also direct teams to focus on efforts that provide the most "bang for the buck," as exemplified by the 80/20 rule, also called the Pareto principle or the "vital few and trivial many." This rule states that approximately 20 percent of the vital tasks produce 80 percent of the results. In the workshops, we apply this principle at week eight, when the final 20 percent of the development effort can make or break the game and outweigh the other 80 percent to bring the game to beta functionality. Besides the obvious addition of visual effects such as shaders and particle systems, students learn to focus on adding game modes and reusing assets.

Visual effects can provide significant enhancements with relatively little effort; likewise, game modes can enhance gameplay with minimal code changes. For example, in the castle defense game, testing revealed that players enjoyed facing attacks by unlimited numbers of trolls until the player's character is killed, in contrast to the normal game mode, in which each level had a set number of attackers. Additional variations include adding a timer to regulate gameplay time or adding defenses that are triggered automatically or by the player.

Simple repurposing of assets can also enhance games. Animated characters can be integrated with menu screens (see Figure 5) or visually augment intralevel story elements. A spherical camera path around a victorious character can enhance a



UP = Jump
DOWN = Fall Faster
LEFT = Move Left
Right = Move Right

SPACE BAR = Attack
Time your hits carefully!

Hold up and a directional button to wall ride. Press the opposite direction to wall jump.

Collect:
Blue Icon = Points
Green Icon = Life

**(a)** **(b)**

**Figure 5. The Project Bolt menu. (a) The main character, Bolt, repurposed on the main menu screen. (b) Bolt outfitted with props for the instruction menu screen.**

win condition. Such asset reuse requires minimal effort but can make the game look much more professional.

## Balancing Structure and Freedom

Without an imposed structure, students' design and development efforts can be misguided owing to their lack of real-world experience. A course can't teach strategy, soft skills, and agile development without imposing the structure to support that teaching. With this additional structure, the quality and complexity of our students' games have increased.

One drawback of this approach is that some students complain about a lack of individual freedom. Although this problem is common in large corporate projects, academic settings require a balance. Requiring formalism can sometimes stifle creativity and enthusiasm, but when we simply recommend a structure, teams tend to ignore it. For example, one team produced a proof-of-concept mod in week three instead of a true new game build; this resulted in late-term integration problems.

Similarly, the agile development methodology and soft-skills training alone can't fully simulate industry conditions because team leaders don't have the true authority that might exist in a corporate environment. This sometimes leads to conflicts, but "employment contracts" for both team leaders and members can help clarify their roles, tasks, and expectations.

Overall, the workshops have successfully prepared our students for industry, as evidenced by Drexel's number-three national ranking in game design (www.games.drexel.edu) and a pairing with EA Games in a *CBS Evening News* segment on preparing students for the industry. Over the past few years, we've made numerous changes to the workshops' structure and the curriculum as a whole. However, the workshops still need continual refinement to balance industry requirements with the exploratory nature of the academic setting.

## References

1. M. Sakey, "Fundamental Learning GDC 2006 Curriculum Workshop," Int'l Game Developers Assoc., Apr. 2006; www.igda.org/articles/msakey_gdc-curriculum.
2. P. Diefenbach, "Teaching Soft-Skills: Digital Game Development in a Multi-discipline Environment," *Eurographics 2008 Annex to the Conf. Proc.*, Eurographics Assoc., 2008, pp. 135–139.

***Paul J. Diefenbach*** *is an associate professor in Drexel University's Digital Media program. Contact him at pjdief@drexel.edu.*

*Contact department editors Gitta Domik at domik@uni-paderborn.de and Scott Owen at sowen@gsu.edu.*