ICE

Intro

ICE je open-source vertikálny shooter vyvíjaný v rámci predmetu Softvérový Projekt na Prírodovedeckej Fakulte UPJŠ v Košiciach.

2D vertikálny shooter je okrem občasnej jednodolárovej mobilnej hry dnes už prakticky mŕtvy žáner. ICE sa vracia ku klasikám ako Raptor a Tyrian a snaží sa žáner posunúť inovatívnym bojovým systémom založeným na RPG prvkoch a využitím moderného hardware. Hrateľnosť je veľmi rýchla a na rozdiel od mnohých dnešných hier nevychádza z predpokladu že hráč je idiot. ICE má jednoduchú vektorovú grafiku v štýle Darwinie alebo DEFCONu.

ICE má veľmi dobrú podporu pre modding. Všetky levely, jednotky a grafika sú v otvorených formátoch a dajú sa jednoducho modifikovať. Postupnou modifikáciou sa dá z ICE urobiť úplne iná hra bez zmeny kódu. A keby z nejakého dôvodu at to bolo potrebné, kód je kompletne otvorený.

ICE momentálne podporuje Windows a Linux, a podpora pre MacOS X nie je ďaleko.

Motivácia

Jednou z obrovských výhod PC hernej komunity ktorú dnešný AAA herný priemysel väčšinou ignoruje je modding. Hry ktoré umožňujú hráčom meniť grafiku, levey či jednotky často žijú desiatky rokov po vydaní, a najlepšie mody prekonávajú hry ktoré boli použité ako ich základ. Mnoho dnešných AAA vývojárov začalo práve modmi pre hry ako Half-Life alebo UT2004.

Cieľom ICE je nie len vytvoriť decentný vertikálny shooter - tých sa nájde obrovské množstvo v rôznych app-marketoch a na abandonware stránkach - namiesto toho sa snažíme vytvoriť platformu pre modderov, na ktorej by sa dala postaviť skoro akákoľvek hra tohoto žánru. Aj samotná hra ICE je iba mod bežiaci pod enginom ICE - všetko sa dá vymeniť alebo modifikovať. ICE by malo slúžiť ako prototyp pre vývoj modderskych technológii, skúsenosti z ktorého sa budú dať využiť pre vývoj iných platforiem pre mody.

ICE nie je prvý projekt svojho druhu - mnoho inšpirácie sme si zobrali z open source ťahovej stratégie *Battle for Wesnoth*, ktorá je centrom obrovskej modderskej komunity. Na rozdiel od Wesnothu má ale ICE byť priateľskejšie k neprogramátorom, podobne ako hra *Red Alert 2* ktorú napriek katastrofálne implementovanému enginu už 12 rokov udržuje pri živote aktívna modderská komunita.

Tým

- Ferdinand Majerech
    - Design a programovanie enginu
    - Organizácia
    - Level design
- Libor Mališ
    - Level design
    - Legacy testing
- Dávid Horváth
    - Level design
    - Windows testing

Technológie

ICE engine bol od začiatku navrnutý pre platformovú nezávislosť. Prístup "ostatné platformy pridáme neskôr" končí katastrofou keď to nakoniec treba rozbehať na inom OS a ukáže sa že celý kód je prerastený platformovo-špecifickými API a bolo by ľahšie to celé prepísať.

Preto sme si zvolili technológie ktoré fungujú na viacerých platformách, a aj tie, pokiaľ možno, sú abstrahované za vnútorné API aby sa dali ľahko vymeniť.

Engine je napísaný v programovacom jazyku *D*, ktorý je platformovo nezávivislý, umožňuje rýchle prototypovanie a pri tom generuje kód dostatočne rýchly pre hry v reálnom čase.

Na vykresľovanie grafiky je použité rozhranie *OpenGL*, ktoré je de-facto štandardným rozhraním pre používanie grafických kariet na skoro všetkých platformách (Výnimkami sú XBox a Windows Phone).

GPU shadery boli programované v GLSL, čo je jazyk dostupný na každej implementácii OpenGL2+.

Na návrh herných entít a levelov sme použili jazyk *YAML*, ktorý je navrhnutý na čo najlepšiu čitateľnosť pre ľudí (narozdiel od napr. XML) a pri tom sa dá rýchlo programovo parsovať. Vysoká čitateľnosť bola prioritou keďže ICE má byť ľahko moddovateľný a hra má byť tvorená predovšetkým dátami, nie enginom samotným.

Fonty sú v ICE vykresľované knižnicou *FreeType*, ktorá je štandardom na vykresľovanie fontov v open source komunite a na väčšine nových platforiem.

Na vstup z klávesnice a prácu s operačným systémom sme použili knižnicu SDL, ktorá podporuje desiatky operačných systémov a je známa svojou stabilitou.

Architektúra

ICE je veľmi striktne rozdelené na engine a hru (alebo ináč povedané mod). Engine obsahuje technickú stránku hry - kód. Hra sa skladá z levelov, entít, grafiky a iných dát. Hra je uložená v adresári - zmenou adresára môže engine načítať inú hru.

Engine

Engine ICE sa skladá z niekoľkých subsystémov so svojími vlastnými API, ktorých spoločným cieľom je načítať herné dáta a vytvoriť z nich hru. Tieto subsystémy sa dajú ďalej rozdeliť na engine systémy herné systémy.

Engine systémy pracujú väčšinou samostatne a tvoria technický základ pre hru (napr. práca s OS, grafika, virtuálny súborový systém). Tieto systémy majú napevno dané miesto v štruktúre kódu a bez nich by engine nemohol fungovať.

Herné systémy sa starajú o hernú logiku. V skutočnosti sú to podsystémy systému entít a všetky majú spoločné API cez ktoré ich systém entít spravuje. Tieto systémy niesú schopné navzájom priamo komunikovať a dajú sa pridávať, odoberať a meniť za behu bez znefunkčnenia hry. Medzi tieto systémy patrí napr. systém zbraní, zdravia, atď.

Engine systémy

- **ICE** sa stará o inicializáciu a deinicializáciu ostatných subsystémov a o vytvorenie hry. Po vytvorení hry v ICE beží "vrchná časť" hlavného cyklu udalostí, kde ICE sleduje či hra neskončila, či sa nestala nejaká chyba, atď.
- **Game** spravuje globálny herný stav, ako sú hráči, bežiaci level a systém entít. Obsahuje logiku na začiatok a ukončenie hry. Obsahuje "spodnú časť" hlavného cyklu udalostí a rozdeľuje čas medzi herné systémy a video systém. (Herné systémy musia mať stabilný počet obnovení za sekundu bez ohľadu na FPS - video systém využije akýkoľvek zostávajúci čas na vykresľovanie.)

- **Platform** je API na komunikáciu s operačným systémom. Jediná implementácia je založená na SDL. Stará sa o vstup z klávesnice a myši, okno v ktorom hra je a grafický mód.
- **VideoDriver** vykresľuje grafiku. Je to veľmi high-level API obsahujúce funkcie na vykreslenie čiar, textu, spritov, rôzne grafické nastavenia atď. Jediná implementácia je založená na OpenGL 2.0 a sama o sebe obsahuje množstvo subsystémov (napr. na virtualizáciu textúr, cachovanie vykresľovacích príkazov a prácu s fontmi).
- **VFS** je virtuálny súborový systém. Dokáže transparentne čítať/písať súbory z/do akéhokoľvek zdroja ktorý má implemetovaný backend. Momentálne to je iba súborový systém OS, ale umožňuje to v budúcnosti napríklad čítať súbory z archívov alebo z FTP. Hlavnou funkciou ktorú momentálne VFS plní je stackovanie adresárov. VFS dokáže pracovať s dvoma (troma, atď.) adresármi ako s jedným adresárom, pričom súbory z neskoršie pridaných adresárov overridujú súbory s predošlých. Toto umožňuje modifikáciu hry bez zmeny pôvodných súborov, jednoduché patche, mody modov, atď.
- **GUI** zobrazuje widgety. Tento subsystém je relikviou dávnych čias a bude nahradený úplne iným GUI systémom.
- **Monitor** sleduje ostatné systémy a má jednoduché GUI na zobrazovanie ich činnosti v reálnom čase. Ostatné systémy majú svoje *submonitory* ktoré sú registrované v Monitore. Monitor sa dá použiť napríklad na zobrazovanie grafov FPS, počtu rôznych vykresľovacích volaní na frame, počtu herných entít, na schematickú mapu detekcie kolízií atď.
- **EntitySystem** spravuje všetky herné entity, a sprístupňuje ich herným systémom. Tento systém obsahuje všetky detaily objektového systému založeného na komponenoch. Stará sa o vytváranie, život a deštrukciu, a garbage collection entít.

Entity

Aby sa herné objekty všetkých druhov dali ľahko definovať bez zásahov do kódu, bolo treba vytvoriť nový, pseudo-dynamický objektový systém. Klasické OOP v takýchto situáciach vedie k obrovským triedam pokrývajúcim všetku možnú funkcionalitu herných objektov (viď Unreal Engine 2). Alternatívou sú špecializované triedy ktorých funkcionalita sá ale nedá kombinovať a dochádza k "pretekaniu" implementácie z kódu do herných dát.

Herné objekty v ICE sú označované ako *Entity* a sú to v skutočnosti len surové dáta bez funkcionality na ktorých operujú herné systémy. Entita sa skladá z jedinečného ID a niekoľkých *komponentov*, ktoré definujú jej vlastnosti. Komponenty ktoré má entita sú definované v YAML a po vytvorení entity sa nedajú meniť. Entity môžu ale vytvárať iné entity (z iných súborov) a meniť ich komponenty, aj rekurzívne.

Príklad entity:

```
visual: visual/arrow.yaml
engine:
  maxSpeed: 600
volume:
  aabbox:
    min: [-20, -30]
    max: [20, 7]
weapon:
  0: weapons/laser.yaml
  1: weapons/mediumPlasma.yaml
collidable:
health: 120
warhead:
  damage: 80
  killsEntity: false
dumbScript: dumbscripts/arrows.yaml
```

Toto je ukážka YAML definujúceho jednoduchú entitu, v tomto prípade nepriateľskú loď. Táto entita má medzi komponentmi grafickú reprezentáciu (`visual`), motor umožňujúci pohyb, zbrane, zdravie, skript určujúci správanie, atď.

V implementácii systému entít entita neexistuje ani ako celistvý objekt. Komponenty jedného typu sú uložené v dynamickom poli a entita obsahuje len ID a indexy do polí každého typu komponentu ktorý v entite je. Obyčajne ale nie sú využívané ani tieto indexy - keď nejaký herný systém spracúva entity, iteruje cez všetky entity ktoré obsahujú komponenty požadovaných typov (napr. *VisualSystem* iteruje cez *PhysicsComponent* a *VisualComponent*). Systém entít vo vnútri iteruje len cez polia komponentov daných typov a cez entity, čo umožňuje efektívnu prácu s cache, a v budúcnosti prefetch, prípadne paralelizáciu (keď dva systémy iterujú cez rôzne komponenty).

Rôzne obmeny komponentového systému sa v poslednom čase začali používať v množstve AAA herných enginov (napr. Resistance 3, Unreal Engine 4, Final Fantasy XV).

Komponenty

Komponent je jednoduchá dátová štruktúra (v zmysle C/C++ - Plain Old Data) s niekoľkými verejnými dátovými členmi načítanými z YAML. Následne môžu byť tieto dáta menené hernými systémami (napr. pozícia fyzikálneho komponentu sa môže meniť). Komponenty môžu existovať iba ako súčasť entít - napr. ak keď YAML súbor entity odkazuje na iný súbor nejakého komponentu, pri načítaní vznikne komponent v tej entite - neexistuje osobitne.

Komponenty samy o sebe neurčujú vlastnosti alebo funkcionalitu entity - tie závisia od herných systémov ktoré komponenty interpretujú a spracúvajú.

Zoznam terajších komponentov:

- **CollidableComponent** - entita dokáže kolidovať s inými entitami
- **ControllerComponent** - Simuluje klávesnicu/gamepad pre kontrolu hráčom/AI/skriptom.
- **DeathTimeoutComponent** - entita zomrie po uplynutí nejakého času (napr. projektily)
- **DumbscriptComponent** - jednoduchý skript (urob toto, urob tamto, žiadny if, for, atď.)
- **EngineComponent** - mení pohyb entity - pôsobí na ňu silou. Hlavne v kombinácii s Controllerom.
- **HealthComponent** - entita má limitované zdravie (určené v YAML), a dá sa zabiť
- **MovementConstraintComponent** - pohyb entity je obmedzený inou entitou alebo globálne
- **OnDeathComponent** - entita spustí nejakú udalosť keď umrie
- **OwnerComponent** - entita ktorá vlastní túto entitu (napr. zbraň čo vystrelila projektil)
- **PhysicsComponent** - pozícia, rotácia, pohyb entity
- **PlayerComponent** - hráč ktorý vlastní túto entitu
- **SpawnerComponent** - vytvára iné entity
- **StatisticsComponent** - ukladá štatistiky (o zásahoch, zabitiach, atď - pre RPG systém)
- **VisualComponent** - grafická reprezentácia entity
- **VolumeComponent** - tvar/veľkosť entity pre detekciu kolízií
- **WarheadComponent** - ako entita vplýva na entity s ktorými koliduje (napr. projektil)
- **WeaponComponent** - zbraň alebo zbrane entity.

Herné systémy

Každý herný systém sa stará o malú časť hernej logiky. Tieto systémy medzi sebou nekomunikujú a nevedia o sebe, ale môžu modifikovať komponenty entít, čo môže ovplyvniť ostatné herné systémy. Ich kombináciou tak vzniká celá herná logika.

Hlavnou časťou každého herného systému je metóda *update()* ktorá zodpovedá jednému obnoveniu hernej logiky pre všetky entity. Táto metóda iteruje cez všetky entity ktoré majú komponenty nejakých konrétnych typov, a iba cez tie ktoré majú všetky vyžadované komponenty. Toto pripomína databázový select keď vyberáme nejaké stĺpce iba z tých riadkov, kde hodnota daného stĺpca nie je NULL. Každý komponent je nejako spracovaný.

Ak nejaké herné systémy spracúvajú rôzne komponenty, malo by byť možné ich spracovávať paralelne vo viacerých vláknach. Momentálne je však rýchlosť akceptovateľná aj bez toho a je ešte veľa priestoru na iné druhy optimalizácie.

Nasleduje výčet terajších herných systémov:

- **CollisionSystem** - detekuje kolízie
- **CollisionResponseSystem** - reaguje na kolízie
- **ControllerSystem** - umožňuje hráčom, AI a skriptom kontrolovať entity
- **EngineSystem** - motory (poháňajú entity)
- **HealthSystem** - zdravie a smrť entít ktorých zdravie klesne na 0
- **MovementConstraintSystem** - obmedzuje pohyby entít vzhľadom na iné entity alebo mapu
- **OnDeathSystem** - aktivuje udalosti pri smrti entít
- **PhysicsSystem** - herná fyzika (nie kolízie, ale napr. pohyb)
- **SpatialSystem** - sleduje v ktorej časti priestoru sa nachádza ktorá entita
- **SpawnerSystem** - vytvára nové entity (napr. v súčinnosti so zbraňami)
- **TimeoutSystem** - spúšťa udalosti v závislosti na čase
- **VisualSystem** - zobrazuje entity (cez VideoDriver)
- **WarheadSystem** - poškodenie a udalosti po kolízii medzi entitami (projektily)
- **WeaponSystem** - logika zbraní, napr. dávky, nabíjanie, atď.

Hra

Hra (herné dáta) sa skladá z jedného virtuálneho adresára (ktorý môže obsahovať viac stackovaných adresárov) obsahujúceho herné súbory. Tento momentálne tvoria adresáre `data/main` a `user_data/main` v adresári ICE.

Medzi hernými súbormi sú, okrem iného, levely, entity a niektoré komponenty (napríklad zbrane sú definované v separátnych súboroch namiesto priamo v súbore entity, aby sa skrátil zápis a aby sa dali zdieľať medzi entitami).

Všetky tieto súbory obsahujú VFS adresy ostatných súborov od ktorých závisia. Takto si modder môže definovať akúkoľvek hierarchiu aresárov, keďže nie sú napevno dané napríklad adresáre pre grafiku, zbrane, atď. Jediný napevno daný "vstupný bod" je momentálne herný level (`levels/level1.yaml`), a v budúcnosti to bude kampaň odkazujúca na levely, alebo možno zoznam kampaní odkazujúci na kampane.

Momentálne sú podľa druhov komponentov a entít adresáre rozdelené nasledovne:

- **dumbscripts** - Skripty určujúce správanie entít
- **explosions** - Explózie entít (entity)

- **fonts** - Fonty
- **levels** - Herné levely (momentálne napevno len `level1.yaml`)
- **logs** - Logy hry (napr. o využívaní pamäte)
- **projectiles** - Projektily (entity)
- **screenshoty** - Screenshoty
- **ships** - Lode (entity)
- **visual** - Vizuálne komponenty entít
- **weapons** - Zbrane

Vytváranie a úprava herného obsahu

Grafika je vektorová, a grafiky jednotlivých jednotiek sú samostatné subory, čo znamená ze je mozné použiť jeden grafický dizajn na viaceré jednotky. Všetka grafika sa skladá z dvojíc bodov, ktoré charakterizujú čiaru pričom pre každú čiaru je možné nastaviť hrúbku pre každý bod farbu, pričom farba postupne prechádza z prvého do druhého bodu.

Zbrane majú rôzne vlastnosti, napríklad čas potrebný na výstrel, počet výstelov, doba nabijania a samostatný opis výstrelu zo zbrane, ktorý sa skladá z nábojov v 1 výstrele - mena súboru ich entity, opisu pozície daných nábojov, nasmerovania a meškania v danom výstrele. Vytvorenie viacerých typov nábojov s rôznym správaním a grafikou umožňuje vytváranie nových zbraní využitím pôvodných nábojov s iným opisom vystrelu.

Definícia jednotky, projektilu, atď. sa skladá z komponentov (zbrane, grafika jednotky, zdravie, atď.). Vytvorenie jednotiek je nevyhnutné k vytvoreniu levelu. Každá entita môže mať definované správanie, teda nie len jednotka, ale napríklad aj náboj. Každé správanie - dočasne pracovne nazvané "dumbscript" je samostatný súbor. V každom správaní je definovaných viacero časových intervalov a pre každý interval je určené správanie - smer pohybu, zmena rýchlosti pohybu, streľba.

Po definovaní správaní môžme definovať level. Každý level sa skladá z niekoľkých vĺn. Každá vlna sa skladá z niekoľkých jednotiek, pričom pre každú jednotku je potrebné určiť cestu k súboru kde sa nachádza opis jednotky, následne pozíciu a natočenie v priestore a je možné určiť správanie jednotky a tak prepísať správanie jednotky ktoré može byť určené už v opise jednotky. Po definícii vlny už nie je problém definovať samotný level, ktorý sa skladá z vĺn a časových páuz medzi vlnami, prípadne nejaký informačný text pre hráča.

Terajší stav

Väčšina enginu je dokončená, chýba ale zvukový subsystém a podpora kampaní (momentálne sa načítava len jeden level zo súboru z napevno napísaným menom súboru). Rýchlosť kódu ešte nie je ideálna (a niektoré framy predchádza niekoľko milisekundový lag, čo je nepríjemné pre hráča.

Vo forkoch členov týmu sú testovacie levely, ktoré sú pomerne hrateľné ale ešte nie dosť vybalancované. Taktiež je už dokončené veľké množstvo entít (nepriatelia, projektily, atď), ktoré sa dajú použiť na tvorbu ďalších levelov.

Ďalší vývoj

Vývoj ICE bude pokračovať behom budúceho semestra, keď by sme sa chceli dostať k prvej releasu-hodnej verzii.

Na strane enginu bude nutné implementovať podporu zvukov a kampaní, a RPG systém pre hráča.

Viac práce ostáva na hre samotnej - bude treba dokončiť príbeh, vytvoriť alebo nájsť použiteľné open-source zvuky a hudbu, a vytvoriť samotnú kampaň. Práca na kampani bude ale tentokrát môcť začať

ihneď narozdiel od tohto semestra, keď bolo treba čakať na engine.

Tiež bude treba vytvoriť stránku pre používateľov (teda nie github) kde by sa dala hra stiahnuť a prezentovať.