Prototyp silnika gry 2D stworzony we frameworku QT

Dawid Kotarba

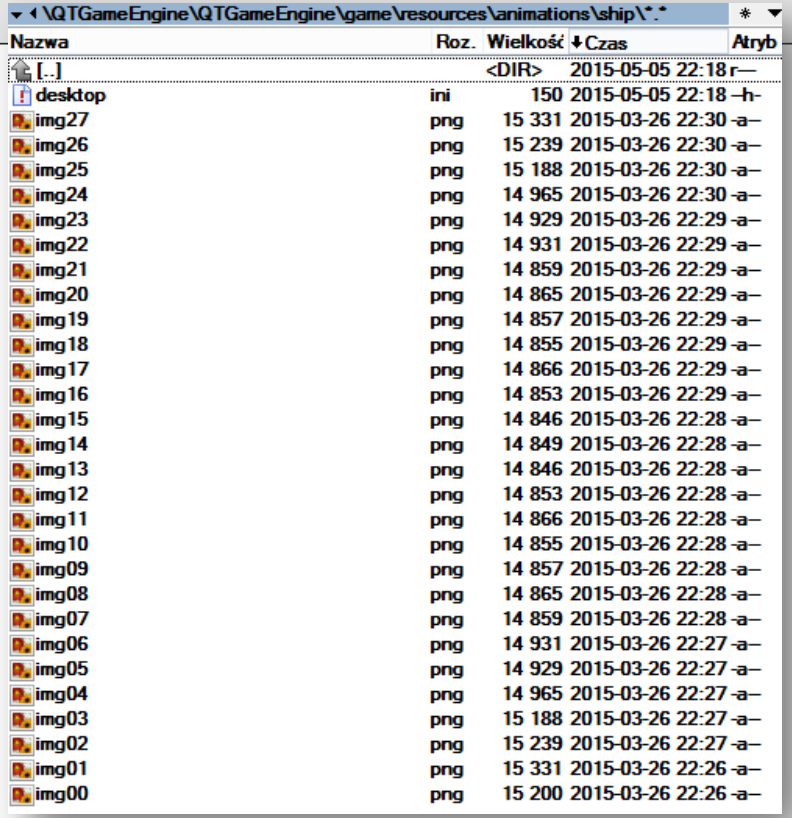
Dlaczego silnik gry w QT?

- C++ idealny do gamedevu
- Duże możliwości frameworka QT:
  - mechanizm sygnałów i slotów
  - implementacja obsługi grafiki 2D [enkapsulacja OpenGL]
  - kolekcje, smart pointery itp.
  - przenośność
  - wspomaganie zarządzania pamięcią
- Całkiem dobre IDE (QT Creator)
- Bogata dokumentacja

Praca magisterska – spis treści

Zautomatyzowane wczytywanie zasobów graficznych

```
1.  Asset(const QString& assetPath);
2.  Asset(const QString& assetPath, int assetImgCount);
3.  Asset(const QString& assetPath, const QString& assetImgPattern, const QString&
    assetImgExtension, int assetImgCount);
```



Zarządzanie zasobami – cache

```
1.  QHash<QString, QPointer<CacheObject<Image> > > imageCache;
```

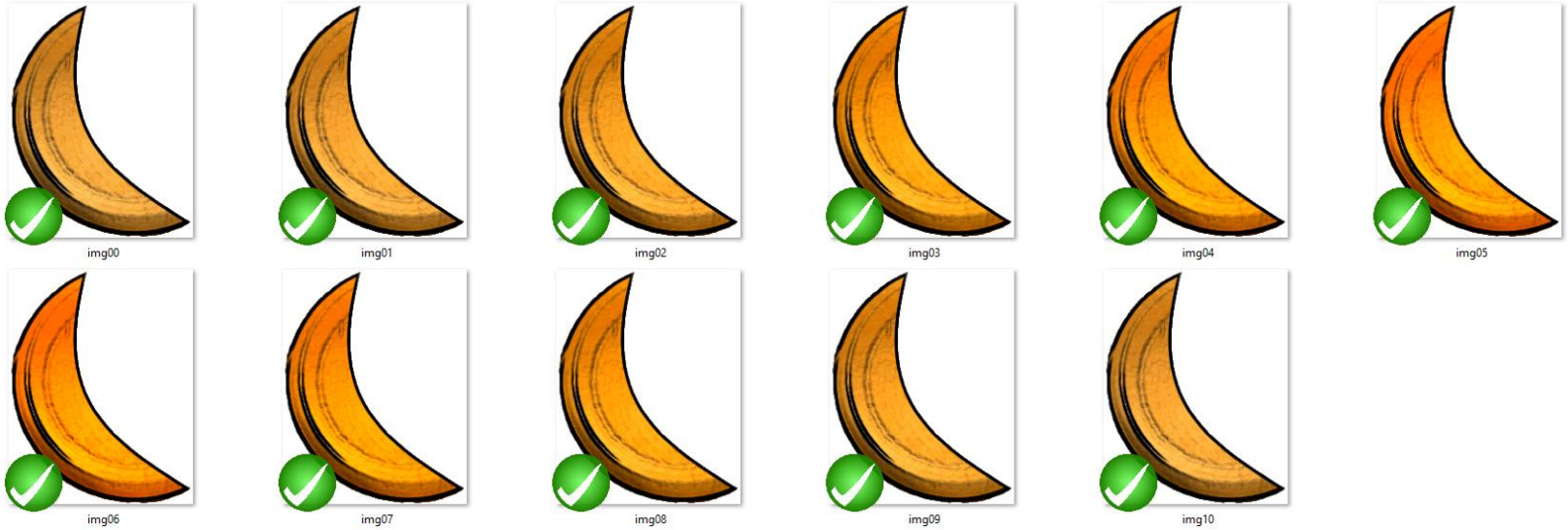
```
1.      void addToCache(QPointer<CacheObject<Image> > item){
2.          imageCache.insert(item->getName(), item);
3.      }
4.
5.      QPointer<CacheObject<Image> > getFromCache(const QString& itemPath){
6.          return imageCache.value(itemPath);
7.      }
```

```
1.  CacheManager::getInstance().addToCache(new
        CacheObject<Image>(imgPathPattern, images));
```

```
1.  QPointer<CacheObject<Image> > cachedImages =
    CacheManager::getInstance().getFromCache(imgPathPattern);
```

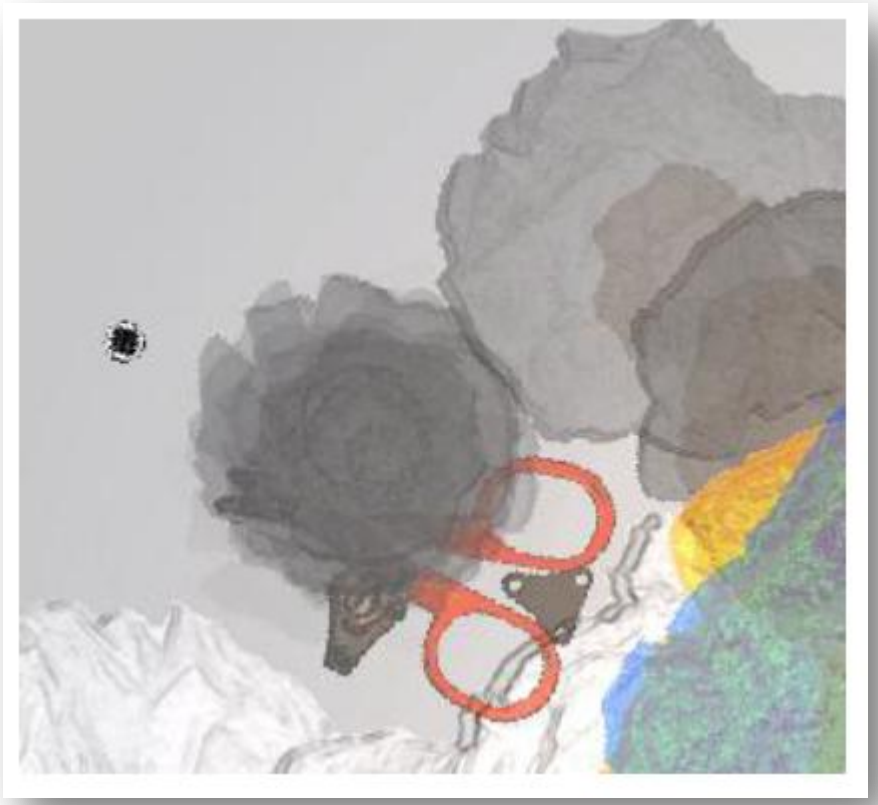
```
1.  template <class T>
2.  class CacheObject: public QObject {
3.  public:
4.      CacheObject(QString path, QList<QPointer<T> > items){
5.          this->path = path;
6.          this->items = items;
7.      }
8.
9.      virtual ~CacheObject(){
10.         clear_qptr_list(items);
11.     }
12.
13.     QString& getName(){
14.         return path;
15.     }
16.
17.     QList<QPointer<T> > getItems(){
18.         return items;
19.     }
20.
21. private:
22.     QList<QPointer<T> > items;
23.     QString path;
24. };
```

Animacje poklatkowe



Transformacje

- Rotacji
- Skalowania
- Zanikania



```
1.  void ItemEffect::rotate(){
2.      if (TimerUtils::getInstance().countEachFrame(frameDelayValue)){
3.
4.          qreal rotation = owner->rotation();
5.
6.          if (rotation > 360)
7.              rotation-=360;
8.
9.          owner->setRotation(rotation+effectFactorValue);
10.     }
11. }
```

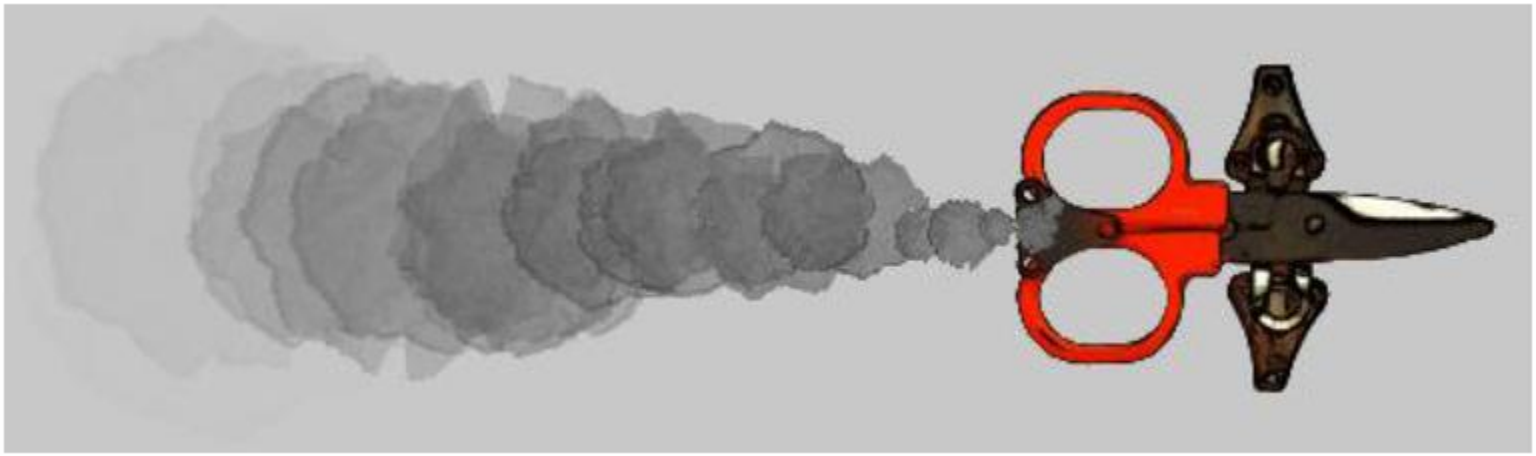
```
12.
13. void ItemEffect::fadeAway(){
14.
15.     qreal opacity = owner->opacity();
16.
17.     if (opacity > 0 && TimerUtils::getInstance().countEachFrame(frameDelayValue)){
18.         owner->setOpacity(opacity-=effectFactorValue);
19.
20.         if (opacity <= 0)
21.             owner->hide();
22.     }
23. }
24.
25. void ItemEffect::scale(){
26.
27.     qreal scale = owner->scale();
28.
29.     if (scale > 0 && TimerUtils::getInstance().countEachFrame(frameDelayValue)){
30.         owner->setScale(scale+=effectFactorValue);
31.
32.         if (scale <= 0)
33.             owner->hide();
34.     }
35. }
```

Detekcja kolizji

```
1.  virtual QRectF boundingRect() const = 0;
2.  virtual QPainterPath shape() const;
3.  QList<QGraphicsItem *> collidingItems(Qt::ItemSelectionMode mode =
    Qt::IntersectsItemShape) const;
```

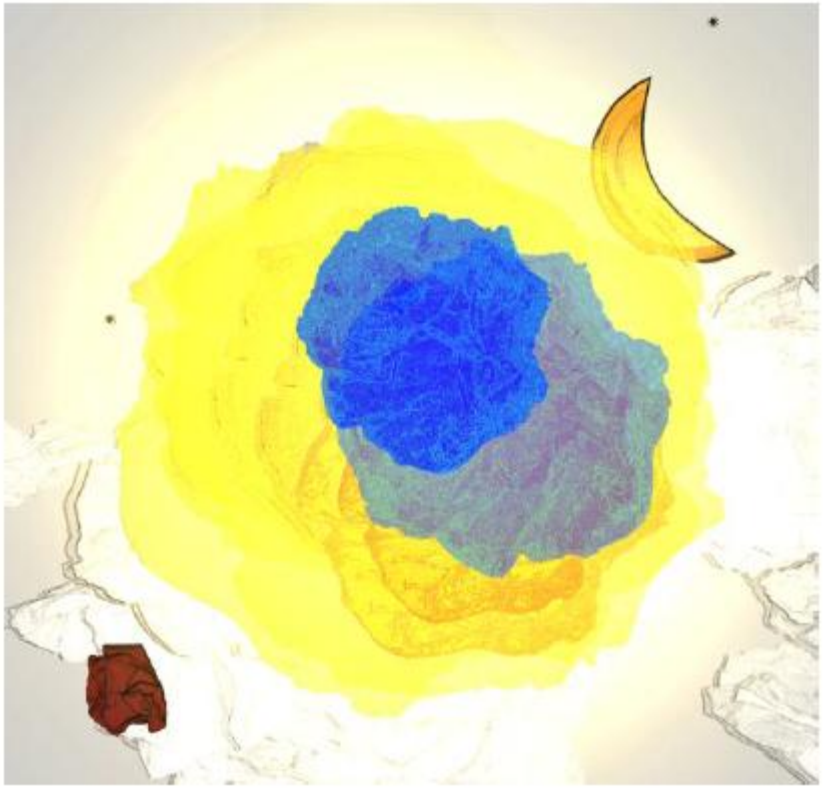
```
1.  QList<QPointer<Item> > collisions = getCollidingItems();
2.
3.      if (!collisions.isEmpty())
4.      foreach (QPointer<Item> col, collisions)
5.          if (col->isDestroyable() && !col->isOutOfScene()){
6.              col->decreaseHealth(damage);
7.              QPoint recoil = WEAPON_DEFAULT_ENEMY_RECOIL;
8.              col->addSpeed(recoil);
9.              die();
10.             return;
11.         }
```

Efekty cząsteczkowe



```
1.     QPointer<ParticlesProcessor> fireball =
2.     new ParticlesProcessor(Asset(fireBallPath1),2);
3.     fireball->getItemsModifier()->setOffset(offset, offset);
4.     fireball->getItemsModifier()->applyRotateEffect(4,5,true);
5.     fireball->getItemsModifier()->setDefaultScale(0.1);
6.     fireball->getItemsModifier()->applyScaleEffect(0.04, 0.06);
7.     fireball->getItemsModifier()->applyFadeEffect(0.02, 0.05);
8.     fireball->getItemsModifier()->moveEveryFrame(const_cast<QPointF&>(MOVE_LEFT_1));
9.     fireball->getItemsModifier()->addLightEffect(500,250,250);
10.    fireball->setLooping(false);
11.    fireball->setRadius(50);
12.    fireball->start();
```

Efekty graficzne: świetlne, cienie, rozmycia



```
1.  void ItemEffect::paintLightEffect(QPainter *painter){
2.      if (effectType != ItemEffectType(LIGHT) || painter == NULL)
3.          return;
4.
5.      QPoint updatedPos(owner->x()+offsetX, owner->y()+offsetY);
6.
7.      painter->setCompositionMode(QPainter::CompositionMode_Plus);
8.      painter->setPen(Qt::NoPen);
9.
10.     QRadialGradient light(updatedPos, effectFactorValue, updatedPos);
11.
12.     updateInnerLightColor(light);
13.
14.     if (lightColor.isValid())
15.         updateOuterLightColor(light, lightColor);
16.     else
17.         updateOuterLightColor(light, COLOR_LIGHT_YELLOW);
18.
19.     painter->setBrush(light);
20.     painter->drawEllipse(updatedPos,effectFactorValue,effectFactorValue);
21. }
22.
23. void ItemEffect::updateInnerLightColor(QRadialGradient& radialGradient){
24.     radialGradient.setColorAt(1.0f, COLOR_WHITE_TRANSPARENT);
25. }
26.
27. void ItemEffect::updateOuterLightColor(QRadialGradient& radialGradient,const QColor&
    lightColor){
28.     QColor lightColAlpha(lightColor);
29.     lightColAlpha.setAlpha(randomLightAlpha());
30.     radialGradient.setColorAt(0.0f,lightColAlpha);
31. }
32.
33. int ItemEffect::randomLightAlpha(){
34.     return Utils::getInstance().randInt(25,100);
35. }
```

Obsługa dźwięku

```
1.  bool SoundUtils::playSound(QString fileName, int volume){
2.
3.      foreach(QPointer<QMediaPlayer> player, soundPlayers){
4.          if (player->state()!= QMediaPlayer::PlayingState){
5.              play(player, fileName, volume);
6.              return true;
7.          }
8.      };
9.
10.     QPointer<QMediaPlayer> newPlayer = new QMediaPlayer();
11.     play(newPlayer, fileName, volume);
12.     soundPlayers.append(newPlayer);
13.     qDebug() << "SoundPlayers increased to " << soundPlayers.size();
14. }
```

```
1.  bool SoundUtils::play(QPointer<QMediaPlayer> player, const QString& fileName, int
    volume){
2.      if (player){
3.          player->setMedia(QUrl(PATH_SOUNDS + fileName + STRING_EXT_MP3));
4.          player->setVolume(volume);
5.          player->play();
6.          return true;
7.      }
8.      return false;
```

Zarządzanie pamięcią

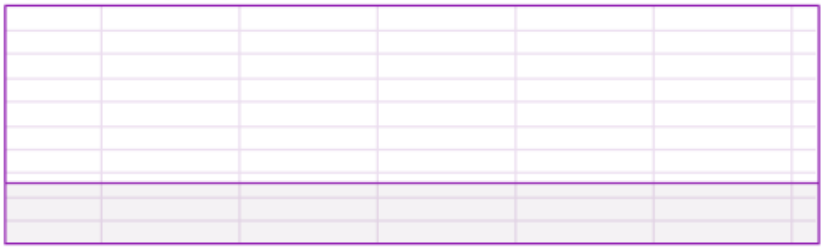
```
1.  template<typename T> void safe_delete(T* pointer){
2.
3.      if (pointer != NULL){
4.          delete pointer;
5.          pointer = NULL;
6.      }
7.  }
8.
9.  template<typename T> void safe_delete_list(QList<T*> list){
10.
11.     foreach (T* item, list) {
12.         safe_delete(item);
13.     }
14. }
15.
16. template<typename T> void clear_qptr(QPointer<T> ptr){
17.     if (ptr != NULL)
18.         ptr.clear();
19. }
20.
21. template<typename T> void clear qptr list(QList<QPointer<T> > list){
22.     foreach(QPointer<T> item, list){
23.         clear_qptr(item);
24.     }
25. }
26. QList<QPointer<QObject> > allItems;
```

```
1.  void MemoryManager::addToItemsList(QPointer<QObject> item){
2.      allItems.append(item);
3.  }
4.
5.  void MemoryManager::clearMemory(){
6.      qDebug() << "Clearing memory manually: " << allItems.size();
7.      clear_qptr_list(allItems);
8.      allItems.clear();
9.  }
10.
11. void MemoryManager::clearMarkedAsForDelete(){
12.
13.     int clearedCount = 0;
14.
15.     for (int i=0; i<allItems.size(); i++){
16.
17.         QPointer<Item> item = (Item*) allItems.at(i).data();
18.
19.         if (!item.isNull() && item->isMarkedAsForDelete()){
20.             clear_qptr(item);
21.             allItems.removeAt(i);
22.             clearedCount++;
23.         }
24.     }
25.
26.     qDebug() << "Cleared items: " << clearedCount
27.     << ". All items: " << allItems.size();
28. }
```

Pamięć

12,0 GB DDR3

Wykorzystanie pamięci 11,9 GB



60 sekund 0

Kompozycja pamięci

W użyciu
2,9 GB

Dostępna
8,8 GB

Zadeklarowana
3,9/13,7 GB

Buforowana
4,9 GB

Pula stronicowana
411 MB

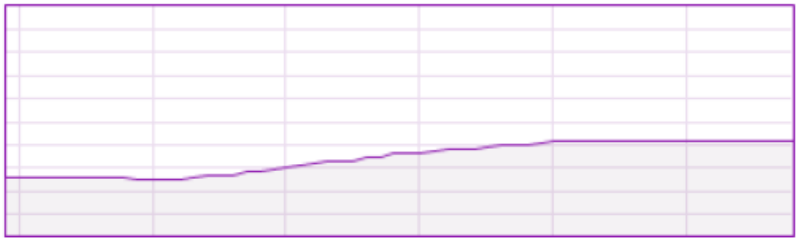
Pula niestronicowana
166 MB

Szybkość:
Użyte gniazda:
Współczynnik postaci:
Sprzętowa zarezerwowana:

Pamięć

12,0 GB DDR3

Wykorzystanie pamięci 11,9 GB



60 sekund 0

Kompozycja pamięci

W użyciu
4,8 GB

Dostępna
6,9 GB

Zadeklarowana
6,8/13,7 GB

Buforowana
4,9 GB

Pula stronicowana
419 MB

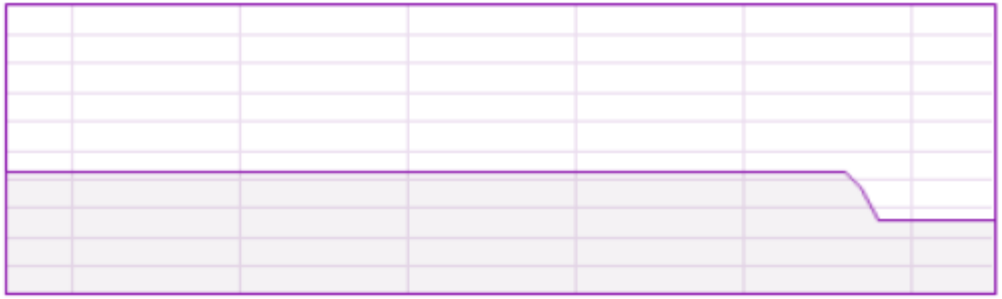
Pula niestronicowana
170 MB

Szybkość:
Użyte gniazda:
Współczynnik postaci:
Sprzętowa zarezerwowana:

Pamięć

12,0 GB DDR3

Wykorzystanie pamięci 11,9 GB



60 sekund 0

Kompozycja pamięci

W użyciu	Dostępna
2,9 GB	8,8 GB
Zadeklarowana	Buforowana
4,0/13,7 GB	4,9 GB
Pula stronicowana	Pula niestronicowana
411 MB	166 MB

Szybkość:
Użyte gniazda:
Współczynnik postaci:
Sprzętowa zarezerwowana:

Generowanie otoczenia i aktorów

```
1.  template <class T>
2.  class ItemSpawner: public QObject {
3.  public:
4.      ItemSpawner(int count, int spreadX, int spreadY, QPoint initialPos =
          QPoint(0,0)):
5.          initialPos(QPoint(0,0)),
6.          itemsModifier(NULL){
7.          this->count = count;
8.          this->spreadX = spreadX;
9.          this->spreadY = spreadY;
10.
11.         for (int itemNo=0; itemNo<count; itemNo++){
12.             QPointer<Item> item = new T();
13.             qreal posX = initialPos.x() + itemNo*spreadX;
14.             qreal posY = initialPos.y()
```

```
1.  QPointer<ItemSpawner<Ufo> > envMoon = new ItemSpawner<Ufo>(params...);
```

```
1.  envStar16 = new ItemSpawner<Star16>(10, 100, 150,
    QPoint(SceneUtils::getInstance().getTranslatedWidth(0),SceneUtils::getInstance().get
    TranslatedHeight(30)));
2.  envStar16->getItemsModifier()->moveEveryFrame(const_cast<QPointF&>(MOVE_LEFT_02));
3.  envStar16->getItemsModifier()->setRepeatable(true);
4.  envStar16->getItemsModifier()->shallBlink(true, 10, 20);
5.  envStar16->start();
```

Prostota API silnika

```
1.  Ufo::Ufo():
2.      Item(Asset(PATH_UFO_BLUE)),
3.      explosionColor(0,0,255){
4.      setDestroyable(true);
5.      setEnemy(true);
6.  }
7.
8.  Ufo::~Ufo(){
9.      clear_qptr(particleExplosion);
10. }
11.
12. void Ufo::move(){
13.     qreal x = this->x() - ENEMY_UFO_SPEED;
14.     qreal y = 0;
15.
16.     if (ENEMY_UFO_SIN_FACTOR != 0)
17.         y = (sin(x/ENEMY_UFO_SIN_FACTOR)*ENEMY_UFO_SIN_FACTOR) + initPos.y();
18.
19.     setPos(x,y);
20. }
21.
22. void Ufo::die(){
23.
24.     particleExplosion = new ParticleExplosion(this, PATH_FIREBALL, PATH_FIREBALL3,
    ENEMY_UFO_PARTICLES_OFFSET, explosionColor);
25.     Item::die();
26.     playSound("chamb");
27. }
```

Wydajność

#	Procesor	Karta graficzna	Ilość pamięci RAM
1.	Intel Core i7 @ 2.3 GHz	Nvidia GeForce GT 650 M	12 GB
2.	AMD A6-6310 @1.8 GHz	AMD Radeon R4	8 GB
3.	Intel Celeron @ 1.5 GHz	Intel HD Graphics 4000	4 GB
4.	Intel Celeron @ 1.3 GHz	Intel HD Graphics 4000	2 GB

Wnioski końcowe

- Prototyp jak najbardziej udany
- Kod generyczny i polimorficzny, łatwy do utrzymania i rozbudowy
- Intuicyjne API
- Banalna budowa gry 2D (losowe generowanie środowiska i aktorów, prostota użycia efektów graficznych)
- Zadowalająca wydajność
- Dobre narzędzie dla osób początkujących w gamedevie

Dziękuję ☺