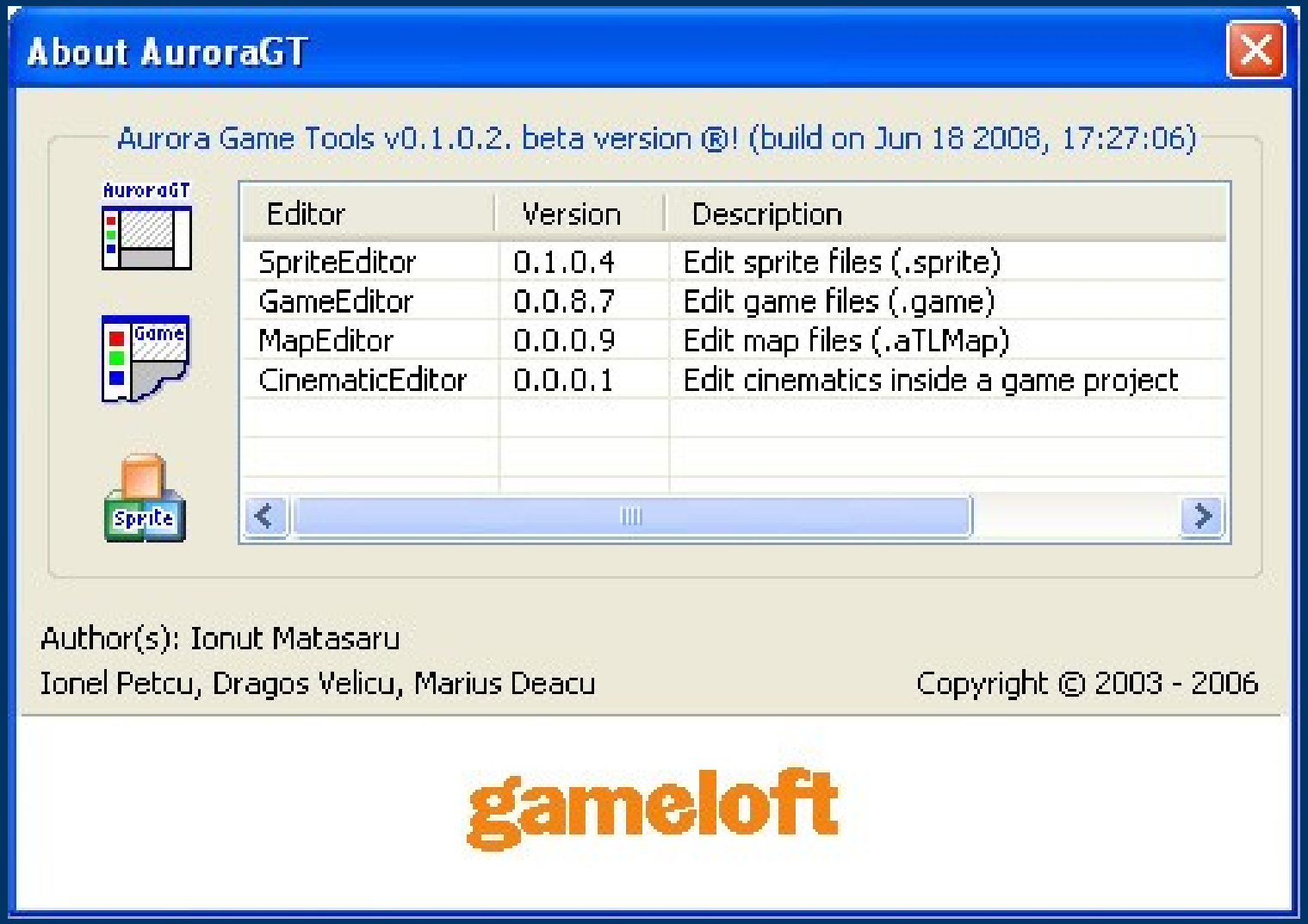
Unleashing AuroraGt

07: Exporting games / Reading binaries

Version

Date	Author	Version	Changelog
٢١/٠٧/٠٨	gaspar.deelias@gameloft.com	١.٠.٥	Initial Version

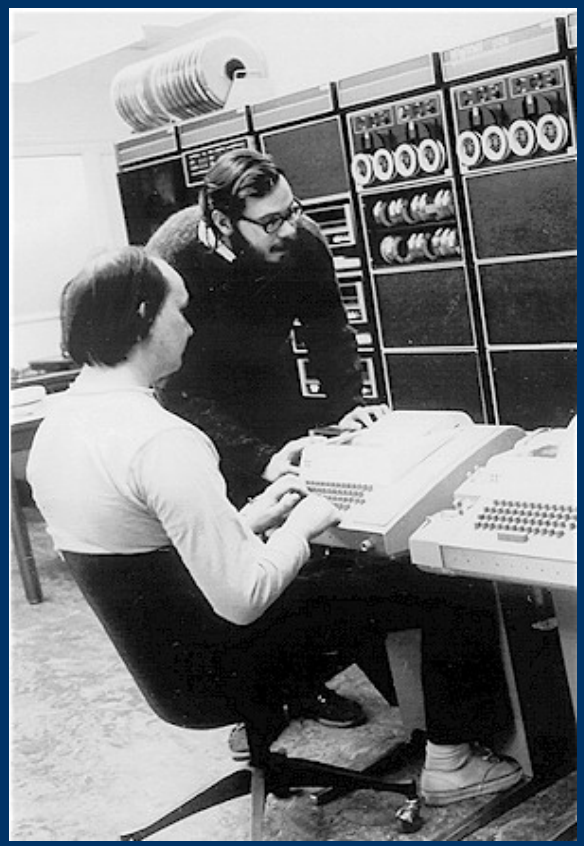
Reference Version[1]



[1] https://terminus.mdc.gameloft.org/vc/tools/AuroraGT (r1189)

Guideline

- In modern computers...
- Resources are almost unlimited...



Dennis Richie and Ken Thompson working on a PDP-11.

- In mobile devices...
- System memory is a delicate thing.
- We cannot just put the bytes

<--Like this one

Guideline

Topics of this presentation:

- Exporting in general.
- .gamecmd syntax.

- FFT Files:
  - For sprites
  - For cinematics

- Binary files: contents.
- Reading the binaries.
- Code examples of the whole loading process.

- After creating a game the following steps are made:
  - Create the export script (.gamecmd file).
  - Create custom format files if needed (.fft file).
  - Run the script to generate the final binaries.
  - Make sure the code can read this information correctly when opening binary files.

AuroraGT

Exporting

- Usually a generic loading code is taken and then adapted for the current game needs.
- Methods such as loadLevel(), loadMap(), etc..
- Exporting in the right format is as important as creating the game!
- If we make a mistake here we won't be able to open level info completely!

AuroraGT

Exporting

- This is a .gamecmd script example:

```
Load("gta-sample.game")

EXPORTGAME
{
  DESC     "Gangstar export"
  PATH     ".\export"
  LOG_FILE  "gta-sample.log"
  MAPS
  {
    MAP
    {
      TEMPLATE     "Ocean_Drive_tiles"
      EXTENSION      "OceanDrive"
      OPTIONS       NO_CONV_TABLE | ENCODE_TILE_FLAGS_I4 | ENCODE_TILESET_MASK_I16
    }
  }

  OBJ_LAYERS
  {
  //  OPTIONS ...
  }
}
```

export.gamecmd example

AuroraGT

Exporting: .gamecmd Structure

- Basic Structure

```
//Use: AuroraGT.exe "file.gamecmd"

Commands:
+ Load("path\in_file.game")
                //-> loads a game into memory
+ EXPORTGAME
{
  DESC     //is the description of the export name
  PATH     //set the current export path the directory where all files are exported
  LOG_FILE  //set the file to log every operation
  MAPS     //MAPS used in game are listed here
  {
      MAP   //define the template that are used
      {
         TEMPLATE "tempName" //use the same name as in .gts file in tiled_layer
         EXTENSION "extension" //binary output will add this next to level name
         OPTIONS //see MAP OPTIONS for further info
      }
      MAP //another map..
      {
         TEMPLATE "tempName2"
         EXTENSION "extension"
         OPTIONS  // OPTIONS
      }
  }
  OBJ_LAYERS  //object from the level
  {
    OPTIONS //OBJ_LAYER OPTIONS
  }
  TASKS
  {
    OPTIONS // TASKS OPTIONS
  }
  STRINGS
}
```

.gamecmd structure

AuroraGT

Exporting: .gamecmd MAP Options

```
OPTIONS //(MAP OPTIONS) can be:
{
  USE_LAYER_NAMES //exported file is named based on layer name
  SIZE_ON_SHORT   //map width and height are exported on short instead of byte
  HEADER_INDICES_ONE_FILE //specify to use other file for map indices

  NO_CONV_TABLE //do not use short array conversion table array
  //( this can be used only to index the tiles from one level).
  //The list contain all tiles that are used in current exported level.
  //This list contain modules frames or anims depend of tile format.
  //Level indices in case this table is used are indexes in that table.

  INDICES_AS_SHORT  //If we have more than 256 tiles
  USE_GAMEWITCH_FLIP_FLAGS  //not used anymore. To keep compatibility with gamewitch flags
  COLLISION //specify that the current map is a collision MAP
  SCEO // this specify the tileset mask name(obsolete,  only for SplinterCell)
  DONT_REMOVE_UNUSED_TILES  //It won't delete unused tiles
  DONT_REMOVE_UNUSED_MODULES  //It won't delete unused modules
  CLONE_FLIPPED_TILES //clone every flipped tile
  REUSE_FLIPPED_TILES //optimization for transformed tiles
  ENCODE_TILE_FLAGS_I4  //Encode each tile in 4 bits. Useful if you have transformations in maps
  ENCODE_TILESET_MASK_I4    //encode tileset mask in I4 format
  ENCODE_TILESET_MASK_I16   //encode tileset mask in I16 format
}
```

.gamecmd: MAP OPTIONS

AuroraGT

Exporting: .gamecmd Object_Layer Options

- These options are applied for every object in the game.
- Map options are defined for every map.

```
OPTIONS //( OBJ_LAYERS OPTIONS) are these:
{
  EXPORT_OBJ_ID //specify to export object unique ID in level.
  //This can be used if you create some kind of link between object in the level

  NOT_EXPORT_OBJ_XY   //do not export object pos x and y
  EXPORT_POINTS_XY    //export points pos x and y
  EXPORT_POINTS_PARAM //export points params
  USE_TEMPLATE_EXPORT_FORMAT  //you can enable custom object layer export as defined
  //in the template section of the GTS file

}
```

.gamecmd: OBJECT_LAYERS OPTIONS

AuroraGT

Exporting

- Have you ever seen an FFT file?
- Do you know what is it for?

AuroraGT

Exporting

- FFT (File Formats Template)
- General use file used to specify AuroraGT export format.
- Used for:
  - Sprites (specify every parameter's datatype)
  - Games(for cinematics, to specify datatypes of default commands (CMD) and new commands (NEW_CMD))
- The scope of this document is the game editor, so we are gonna take a look at FFT files for cinematics.

AuroraGT

FFT: Cinematics 1

- Available formats are:
  - INT8
  - UINT8
  - INT16
  - UINT16
  - INT32
  - UINT32
- These params have influence in the code

```
FILE_FORMAT_TEMPLATE "GameFFT"
{

  CHUNKS
  {
    game.for_each_level
      level.open_cinematics_file
        level.number_of_cinematics : INT8
        level.for_each_cinematic
          cinematic.id : INT16
          cinematic.number_of_tracks : INT8
          cinematic.number_of_key_frames : INT16
          cinematic.for_each_track
            track.type : INT8
            track.switch_by_type
            {
              case CTRACK_BASIC:
                track.flags : INT8
              case CTRACK_CAMERA:
                track.flags : INT8
              case CTRACK_OBJ_LAYER:
                track.flags : INT8
                track.object_layer_id : INT16
              case CTRACK_SI:
                track.flags : INT8
                track.sprite_id : INT16
            }
            track.number_of_key_frames  : INT16
            track.for_each_key_frame
              key_frame.time : INT16
              key_frame.number_of_commands : INT8
              key_frame.for_each_command
                key_frame_cmd.type : INT8
                key_frame_cmd.switch_by_type_ex
                {
                  ////////////////////////////////////////////////////////////
                  // Standard commands...
```

FFT: Cinematics part 1/3

AuroraGT

FFT: Cinematics 2

- Here we can see all default cinematic CMDs
- Each one has its type.

```
key_frame_cmd.switch_by_type_ex
{
  ////////////////////////////////////////////////////
  // Standard commands...

  case "Camera.SetPos.PosX"      : INT16
  case "Camera.SetPos.PosY"      : INT16
  case "Camera.CenterTo.PosX"      : INT16
  case "Camera.CenterTo.PosY"      : INT16
  case "Camera.FocusOn.Thread"     : INT8
  case "Camera.FocusOn.OffsetX"    : INT16
  case "Camera.FocusOn.OffsetY"    : INT16

  case "Basic.SetPos.PosX"       : INT16
  case "Basic.SetPos.PosX"       : INT16
  case "Basic.SetAction.Action"    : INT16
  case "Basic.SendObjEvent.ObjectID"  : INT16
  case "Basic.SendObjEvent.Param"    : INT16
  case "Basic.SendObjEvent2.ObjectID" : INT16
  case "Basic.SendObjEvent2.Param1" : INT16
  case "Basic.SendObjEvent2.Param2" : INT16
  case "Basic.SendObjEvent3.ObjectID" : INT16
  case "Basic.SendObjEvent3.Param1" : INT16
  case "Basic.SendObjEvent3.Param2" : INT16
  case "Basic.SendObjEvent3.Param3" : INT16
  case "Basic.SendEvent.Param"     : INT16
  case "Basic.SendEvent2.Param1"     : INT16
  case "Basic.SendEvent2.Param2"     : INT16
  case "Basic.SendEvent3.Param1"     : INT16
  case "Basic.SendEvent3.Param2"     : INT16
  case "Basic.SendEvent3.Param3"     : INT16

  case "ObjThread.SetPos.PosX"     : INT16
  case "ObjThread.SetPos.PosY"     : INT16
  case "ObjThread.SetAnim.Animation"  : INT8
  case "ObjThread.AddFlags.Flags"    : INT32
  case "ObjThread.RemoveFlags.Flags"  : INT32
```

AuroraGT

FFT: Cinematics 3

- Custom cinematic commands (NEW_CMD)

```
//////////////////////////////////////////////////////
// Custom commands...
case "Basic.HideLayerZone.Rect"    : INT16
case "Basic.HideLayerZone.LayerID"  : INT16
case "Basic.StartDialog.DialogID"      : INT16
case "Basic.StartDialog.ShowedTimeMS" : INT16
case "Basic.SetActorFlags.Actor"     : INT16
case "Basic.SetActorFlags.ActorFlags" : INT16
case "Basic.ResetActorFlags.Actor"     : INT16
case "Basic.ResetActorFlags.ActorFlags" : INT16
case "Basic.StartCountdown.EndAction" : INT16
case "Basic.StartCountdown.TimeInSecs"  : INT16
case "Basic.StartChase.Actor"      : INT16
case "Basic.StopChase.Actor"       : INT16
case "Basic.QueueNextCinematic.NextCinematicUID" : INT16
case "Basic.PlaySound.SoundID"       : INT16
case "Basic.PlaySound.NumLoops"      : INT16
case "Basic.PlaySound.Volume"      : INT16
case "Basic.PlaySound.Priority"      : INT16
case "Basic.SwitchShrekState.NewState"  : INT16
case "Basic.SetCameraBounds.Rect"    : INT16
case "Basic.ChangeHiddenRegionState.Rect"    : INT16
case "Basic.ChangeHiddenRegionState.NewState"    : INT16
case "Basic.StartItemFalling.ItemToFalls" : INT16
case "Basic.StartItemFalling.Direction" : INT16
case "Basic.ModifyObjectAttribute.Target" : INT16
case "Basic.ModifyObjectAttribute.Action" : INT16
case "Basic.ModifyObjectAttribute.Param" : INT16
case "Basic.StartDialogType.DialogID"      : INT16
case "Basic.StartDialogType.ShowedTimeMS" : INT16
case "Basic.StartDialogType.DisplayType"  : INT16
case "Basic.Wait.Type"     : INT16
case "Basic.Wait.Param"    : INT16
case "Basic.SetCameraOffsets.OffsetX" : INT16
case "Basic.SetCameraOffsets.OffsetY" : INT16
case "Basic.ReFocusCameraOnPlayer.FocusType" : INT16
```

AuroraGT

FFT

- Datatypes for NEW_CMDs must be specified here
- Then process the FTT file including these lines in the .gamecmd file:
  - InitGameFFT("game_export.fft")
  - ExportGameFFT(WRITE_LOG)

AuroraGT

Game Binaries and its contents...

- Binaries generated:
  - <levelName>.cinematics // cinematics data
  - <levelName>.layers // objects data (entities in level)
  - <levelName>_<extension>_map1.bin //tiled layers data (aTLMap)
  - <levelName>_<extension>_map1_h.bin // width & height
  - <levelName>_<extension>_map1_flags.bin //tiled layer flags (aTLMap)

- <levelName> Level name in aurora editor.
- <extension> Map extension defined in GTS file.

AuroraGT

Reading Exported Binaries

- This is what we did at the moment:
  - Create sprites and maps
  - Write the game template (.gts)
  - Design levels in aurora (.game)
  - .gamecmd file done
  - .fft done (if needed)

Q: What's next? What do we do with our binaries?

A: We may use them in our game!

- That's the easiest part since everything is already done for many gameloft games!
- Heres the game loading process....

AuroraGT

Game loading process

- createLevel()
  - 1. createBuffer //create an empty image to be used as buffer.
  - 2. loadMap() //this is made for every level layer
    - readTileMap() //tileId is saved in an array for the whole layer
    - readFlipMap() //transformations for every layer tileId
    - readPhisicsMap() //if theres a collision map is loaded!
  - 3. createTileSetImage()
    - createSprites //sprite.loadSprite() call for required tileset.
    - cache tiles... //sprite.BuildCacheImages() call for the tileset.
  - 4. loadLevel()
    - createEntities() //create actors using loadmap() info
    - createCinematics() //save level cinematics info in memory.
  - 5. maskLevelSprites()
    - //read level entities list and mark sprites for load.
  - 6. loadMarkedSprites() //calls to sprite.loadSprite() method.
  - 7. createEntities //create level actors.

AuroraGT

Game loading process: 1. createBuffer()

```
// we create a Buffer at this step
// depending on the game, the buffer size is equal to screen
// size or bigger if buffer is cyclic

Image imgBuff = Image.createImage(SCR_W, SCR_H);
Graphics gBuff;
gBuff = imgBuff.getGraphics();

// now render can be done in gBuff and we will see the
// results in memory and not in the real screen.

//after calculating/rendering actual map....

// g.drawImage(imgBuff, posX, posY); //where g is the screen
```

AuroraGT

Game loading process: 2. loadMap()

```
cGame.pack_open(s_levelPack[level]);

//sky layer, will fill s_mapDataF, s_mapFlipF
s_mapDataF = cGame.ReadTileMap(LEVEL1_F_MAP);//far tilemap
s_mapFlipF = cGame.ReadFlipMap(LEVEL1_F_MAP_FLAG);flags

s_mapFTWidth = s_mapTWidth;
s_mapFTHeight = s_mapTHeight;

// phisics layer (no flips obviously)
s_mapDataP = cGame.ReadTileMap(LEVEL1_P_MAP);

// background layer
s_mapDataB = cGame.ReadTileMap(LEVEL1_B_MAP); //background
s_mapFlipB = cGame.ReadFlipMap(LEVEL1_B_MAP_FLAG);//flags

s_mapWidth = s_mapTWidth * TILE_WIDTH;
s_mapHeight = s_mapTHeight * TILE_HEIGHT;

cGame.pack_close();
```

AuroraGT

Game loading process: 3. createTileSetImage()

```
cGame.pack_open(PACK_TILESET);
// create tileset bottom
s_sprTilesetB = cGame.LoadSprite(s_levelData[(s_level * 2)+0]);
s_sprTilesetF = cGame.LoadSprite(s_levelData[(s_level*2)+1]);

cGame.pack_close();

int pal = s_levelPalette[ (s_level * 2) + 0];
s_sprTilesetB.SetCurrentPalette(pal);
s_sprTilesetB.BuildCacheImages(pal, 0, -1, -1);
s_sprTilesetB.ClearCompressedImageData();

s_imgTilesetB = s_sprTilesetB._modules_image[pal][0];
s_sprTilesetB = null;

int pal = s_levelPalette[ (s_level * 2) + 1];
s_sprTilesetF.SetCurrentPalette(pal);
s_sprTilesetF.BuildCacheImages(pal, 0, -1, -1);
s_sprTilesetF.ClearCompressedImageData();

s_imgTilesetF = s_sprTilesetF._modules_image[pal][0];
s_sprTilesetF = null;
```

AuroraGT

Game loading process: 4. loadLevel()

```
byte[]  data_entities; //here we store every game object
byte[]  data_cinematics; //save cinematics in memory

cGame.pack_open(s_levelPack[level]);

data_entities = cGame.pack_readData(LEVEL1_LAYER);
data_cinematics = cGame.pack_readData(LEVEL1_CINEMATICS);

cGame.pack_close();

s_entities = data_entities;

cGame.LoadCinematics(data_cinematics);
```

AuroraGT

Game loading process: 5. markLevelSprites()

```
int paramNum;
short params[] = new short[MAX_PARAMS]; //id,x,y, & params

while (i < s_entities.length) {
   paramNum = s_entities[i++];

   //read parameters for current entity
   for (int j = 0; j < paramNum; j++)
     params[j] = (short) ((s_entities[i++] & 0xFF) +
     (s_entities[i++] << 8));

   ItemUesdFlag[params[FLAG_IMAGE]] = 1; //flag it for load.
}//end while
```

- Basically this code will read every entity parameter.
- Notice that parameters are stored in shorts

AuroraGT

Game loading process: 6. loadMarkedSprites()

```
// this is made for every "idx"

if (SpriteUesdFlag[idx] == 1)
{
    s_sprites[idx] = cGame.LoadSprite(idx);

    int pal = 0;
    s_sprites[idx].BuildCacheImages(pal, 0, -1, -1);
    s_sprites[idx].ClearCompressedImageData();
}
```

AuroraGT

Game loading process: 7. createEntities()

- Here we create level actors...
- Number of params are stored in byte.
- Param's datatype is short.

```
int paramNum;
short params[] = new short[MAX_PARAMS]; //id,x,y, & params

while (i < s_entities.length)
{
    paramNum = s_entities[i++];
    for (int j = 0; j < paramNum; j++)
        params[j]=(short)((s_entities[i++]&0xFF)+(s_entities[i++] << 8));
    cActor actor = new cActor(params);
    actor.PostConstruct(params);

}
```

AuroraGT

Conclusion

- Get into Asprite class and see it by yourself.
- The Exported data type is really important at the time of reading binaries
- The load sequence may change a little from game to game, but the idea is the same.
- Be curious about this topic and you will master Aurora.

AuroraGT

Bibliography

- **AuroraGT official repository**
  https://terminus.mdc.gameloft.org/vc/tools/AuroraGT
- **AuroraGT main wiki**
  https://wiki.gameloft.org/twiki/bin/view/Main/AuroraGT

AuroraGT

Contact us

- Please, we look forward for any suggestions or bug found:
  - send us a mail to World-AuroraSuggestions@gameloft.com