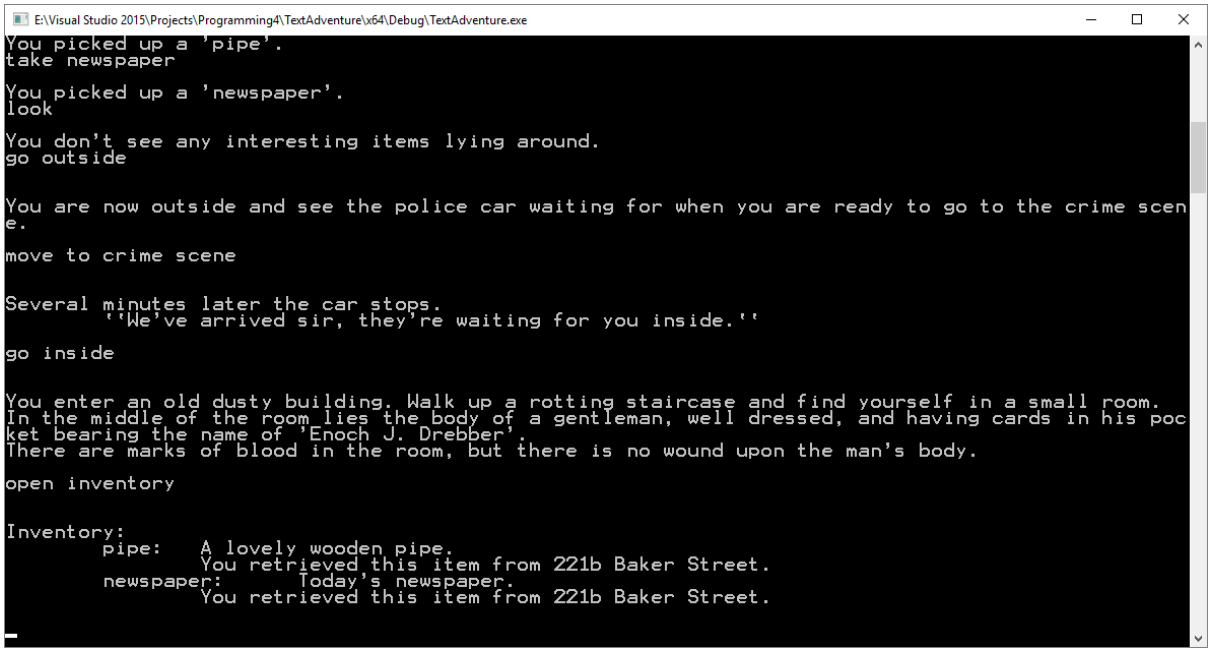
Code samples

Yentoff Cuypers

C++ TextAdventure project with focus on lambdas and regex



Function using an std map with regexes as keys and functions as values for generic function calling.

```
map<string, function<void(const cmatch &)>> m_actionDictionary;
```

```
void Game::ParseInput(const string& input)
{
	//Lambda-ception test
	/*auto lambdalambda = [&](auto pair)
	{
		regex r(pair.first, regex_constants::ECMAScript | regex_constants::icase);
		cmatch base_match;
		if (regex_search(input.c_str(), base_match, r))
		{
			(pair.second)(base_match);
		}
	};
	for_each(m_actionDictionary.begin(), m_actionDictionary.end(), lambdalambda);*/

	for (auto pair : m_actionDictionary)
	{
		regex r(pair.first, regex_constants::ECMAScript | regex_constants::icase);
		cmatch base_match;
		if (regex_search(input.c_str(), base_match, r))
		{
			(pair.second)(base_match);
			break;
		}
	}
}
```


www.yentoffcuypers.com yentoff.cuypers@student.howest.be

Initialization of an action.

```
m_actionDictionary["^(look)( around)?( *)"] = [&](const cmatch & args)
{
	auto items = m_pGameContext->m_pCurrentAreaNode->GetItems();
	if (items.size() <= 0)
	{
		stringstream ss;
		ss << "You don't see any interesting items lying around.\n";
		SlowPrint(ss.str());
	}
	else
	{
		stringstream ss;
		ss << "You look around and notice the following items:\n";
		for (auto item : items)
		{
			auto name = item->GetName();
			ss << "\t" << item->GetName() << ":\t"
			<< (name.size() < 8 ? "\t" : "")
			<< item->GetDescription() << endl;
		}
		ss << endl;
		SlowPrint(ss.str(), 5);
	}
};
```

C++ Kirby Super Star remake

Set timed collision flags.

```
	if (m_HitByEnemyTime >= m_HitByEnemyTimeMax)
	{
		m_IsHitByEnemy = false;
		m_HitByEnemyTime = 0;
		b2Filter filter = m_ActPtr->GetCollisionFilter();
		filter.maskBits |= (uint16)FilterCategory::ENEMY |
(uint16)FilterCategory::TRAP;

		m_ActPtr->SetCollisionFilter(filter);
		m_CollisionFilterUpdated = false;
	}

	if (m_IsHitByEnemy)
	{
		m_HitByEnemyTime += deltaTime;
		if (!m_CollisionFilterUpdated)
		{
			b2Filter filter = m_ActPtr->GetCollisionFilter();
			filter.maskBits &= ~(uint16)FilterCategory::ENEMY &
~(uint16)FilterCategory::TRAP;
			m_ActPtr->SetCollisionFilter(filter);
			m_CollisionFilterUpdated = true;
		}
	}
```

C# (Unity) Prototyping project

Update function for a smooth damped 3rd person camera which rotates around player, separately from the player rotation.

```
void UpdateRotationAndPosition()
    {
        // Rotate camera around the Player
        // Rotation Input
        float hInput = Input.GetAxis("HorizontalTurn");
        float vInput = Input.GetAxis("VerticalTurn");

        // set the direction multipliers:
        // right = -1 -> move camera to right and counter rotate
        //           (to the left, towards player)
        // left = 1 -> move camera to left and counter rotate
        //           (to the right, towards player)
        // up = -1, down = 1 -> same idea.
        float hRotateDirection = 0, vRotateDirection = 0;

        if (hInput < 0) hRotateDirection = 1;        // left
        else if (hInput > 0) hRotateDirection = -1;  // right
        if (vInput < 0) vRotateDirection = 1;        // down
        else if (vInput > 0) vRotateDirection = -1;  // up

        // add new rotation angle to horizontal and vertical angles.
        _camSettings.AngleHorizontal += hRotateDirection *
_camSettings.RotationSpeedHorizontal * _camSettings.Distance * 0.02f;

        _camSettings.AngleVertical -= vRotateDirection *
_camSettings.RotationSpeedVertical * 0.02f;
        _camSettings.AngleVertical = ClampAngle(_camSettings.AngleVertical,
_camSettings.VerticalMinLimit, _camSettings.VerticalMaxLimit);

        Quaternion rotation = Quaternion.Euler(_camSettings.AngleVertical,
_camSettings.AngleHorizontal, 0);

        // calculate distance offset and target position
        Vector3 negDistance = new Vector3(0.0f, 0.0f, -_camSettings.Distance);
        Vector3 position = rotation * negDistance + _camSettings.Target.position;

        transform.rotation = rotation;

        // Smooth damped camera movement to follow Player
        Vector3 velocity = Vector3.zero;
        this.transform.position = Vector3.SmoothDamp(this.transform.position,
position, ref velocity, _camSettings.SmoothDamp);
    }
```

C# (Unity) The Textbook Robbery

Multithreaded function to calculate the average normal used for the outlines.

```
void CalculateOutline(GameObject meshObj)
    {
        var meshFilter = meshObj.GetComponent<MeshFilter>();

        var m = meshFilter.sharedMesh;

        string filePath = EditorUtility.SaveFilePanelInProject("Save TangentNormals Mesh",
                    meshObj.name + "_TN", "asset", "");
        if (filePath == "") return;

        Mesh mesh = meshFilter.mesh;

        AssetDatabase.CreateAsset(mesh, filePath);

        Vector4[] avgNormals = new Vector4[mesh.vertexCount];
        // temp lists because unity doesn't allow direct access of Unity objects
        // outside of its main thread.
        var vertices = mesh.vertices;
        var normals = mesh.normals;

        var thread = new Thread(
          () =>
          {
              #region Calc avg normals
              // Go over all vertices
              for (int i = 0; i < vertices.Length; ++i)
              {
                  // For each vertex, check if there is another vertex with same position,
                  // if found -> add the normal and avg them
                  Vector3 average = normals[i];
                  List<int> foundVerts = new List<int>();
                  for (int j = 0; j < vertices.Length; ++j)
                  {
                      if (i == j) continue;
                      if (vertices[i] == vertices[j])
                      {
                          foundVerts.Add(j);
                          average += normals[j];
                      }
                  }

                  average /= 1 + foundVerts.Count;

                  if (foundVerts.Count > 0)
                  {
                      for (int j = 0; j < foundVerts.Count; ++j)
                      {
                          avgNormals[j] = (Vector4)average;
                      }
                  }

                  avgNormals[i] = (Vector4)average;
              }
              #endregion
          });
        thread.Start();
        thread.Join();

        mesh.tangents = avgNormals;
    }
```

C++ MySQL database access

Function from a simple singleton database connection class using MySQL's cppconn library.

```
sql::PreparedStatement * DatabaseConnection::PrepareStatement(const std::string & query)
{
	return m_Con->prepareStatement(query);
}

int DatabaseConnection::ExecuteUpdate(sql::PreparedStatement * stmt)
{
	int res = 0;
	try
	{
		res = stmt->executeUpdate();
		cout << "Number of rows updated: " << res << endl;
	}
	catch (sql::SQLException &e)
	{
		cout << "# ERR: SQLException in " << __FILE__;
		cout << " ( " << __FUNCTION__ << " ) online" << __LINE__ << endl;
		cout << "# ERR: " << e.what();
		cout << " (MySQL error code : " << e.getErrorCode();
		cout << " , SQLState : " << e.getSQLState() << " ) " << endl;
	}
	return res;
}
```

Implementation function to insert a Player into a Clan Database.

```
bool ClanPlayerDAOImpl::Insert(unsigned int clanId, unsigned int playerId, unsigned int
roleId)
{
	// Is first player to be added to clan -> GuildLeader
	if (GetAllPlayersInClan(clanId).size() <= 0)
	{
		roleId = 1;
	}

	DatabaseConnection * db = DatabaseConnection::GetInstance();

	sql::PreparedStatement * stmt =
		db->PrepareStatement("insert into Clan_Player(clanId, playerId, roleId)
values (?,?,?);");

	stmt->setUInt(1, clanId);
	stmt->setUInt(2, playerId);
	stmt->setUInt(3, roleId);

	int nrOfUpdatedRows = db->ExecuteUpdate(stmt);
	delete stmt;

	return nrOfUpdatedRows > 0;
}
```

Implementation function retrieving all Players in a certain Clan from database.

```
vector<Player> ClanPlayerDAOImpl::GetAllPlayersInClan(unsigned int clanId)
{
	stringstream ss;
	ss << "select p.* from Clan_Player cp inner join Player p on cp.playerId =
p.id where cp.clanId = ?;";

	DatabaseConnection * db = DatabaseConnection::GetInstance();

	sql::PreparedStatement * stmt = db->PrepareStatement(ss.str());

	stmt->setUInt(1, clanId);

	sql::ResultSet *res = stmt->executeQuery();

	std::vector<Player> players;

	while (res->next())
	{
		Player p(res->getUInt("id"), res->getString("name"), res-
>getUInt("level"));

		players.push_back(p);
	}
	delete res;
	delete stmt;

	return players;
}
```

Feel free to contact me for more.