Real-time Adaptive Track Generation in Racing Games

Jake Bird, Tom Feltwell and Grzegorz Cielniak
School of Computer Science
University of Lincoln, UK
email: birdyjake@aol.com; tfeltwell,gcielniak@lincoln.ac.uk

KEYWORDS
Procedural Content Generation, Real-time Adaptability, Racing Car Simulation

ABSTRACT

This paper addresses the problem of adaptability in the context of racing games and proposes a real-time track generation system automatically adjusting to the player's performance. The system modifies the track difficulty according to a heuristic model of the player's experience. The conducted experiments demonstrate the feasibility of real-time adaptability in the racing game context and show that this feature can positively influence the player's perception of challenge and fun.

INTRODUCTION

Player's gaming experience is the most critical aspect of modern game development. The so called "fun" factor captures the essence of that experience and provides intuitive understanding of what is expected from game designers. In context of racing games, the "fun" factor is affected largely by the amount and varied type of challenge, which can be attributed to characteristics such as ability to drive fast, varied tracks (i.e., avoiding long straight sections or continuous bends), realistic simulation of the car, including drifting and skidding, and continuous improvement of skills both for beginner and expert players (Togelius et al. 2006). It is important to note that the player's perception of challenge is not static but dynamically changes together with their improving abilities.

Commercial games are usually not tailored to incorporate any individual playing style, at most providing a fixed set of generic difficulty settings. The lack of *adaptability* is an inherent characteristic of a traditional, manual design process which negatively affects the "fun" factor, re-playability, potential for exploring the game to its full extent, but also associated asset creation costs.

RELATED WORK

Recently, Procedural Content Generation (PCG) techniques have been proposed to address various challenges in video game context (see (Togelius et al. 2011) for an extensive survey). PGC methods are being used during the game development phase to aid designers in automated asset creation, including generation of decorative elements like weather conditions, lighting or textures (e.g., (Whitehead 2010)), but also of terrain, foliage or game levels (e.g., (Hastings et al. 2009, Pedersen et al. 2010)). In such scenarios, the quality of created assets is judged by the designer, who decides if a particular result is suitable for the specific game.

PCG methods can also be used during the actual gameplay, enabling adaptability of various design aspects (e.g. assets, NPCs or game mechanics). In this case, the feedback information to the adaptive algorithm consists of some measure of player's experience, and the general aim is to improve that experience by introducing varying challenge and novelty into the game-play (e.g., (Yannakakis and Togelius 2011)). There are several ways of measuring the player's experience including questionnaire-based feedback, by taking physiological (i.e., biometric) responses or extracted directly from the game-play, by monitoring metrics that approximate player's performance, skill and engagement (Bernhaupt 2010). The former two approaches pose obvious limitations including subjectivity of the questionnaire response or additional apparatus required for measuring biofeedback.

The analysis of in-game generated metrics for adaptation is particularly interesting as it can be applied in real-time. There are however only a few examples of such systems implemented so far, including for example automatically adjusted difficulty in physical games (Yannakakis and Hallam 2009) or FPS games (e.g. Left4Dead (Booth 2009)). In racing context, the adaptation has been explored for modelling AI opponents (e.g. Drivatar system (Microsoft Research 2012)) or adapting tracks through evolutionary algorithms (Togelius et al. 2006). However, adjustment in these systems does not occur in real-time.

This paper presents a system for generating racing tracks which are adjusted in real-time according to the player's performance. The presented experiments demonstrate that the real-time adaptability is feasible in the racing game context and that it can positively influence the player's perception of challenge and fun.

THE METHOD

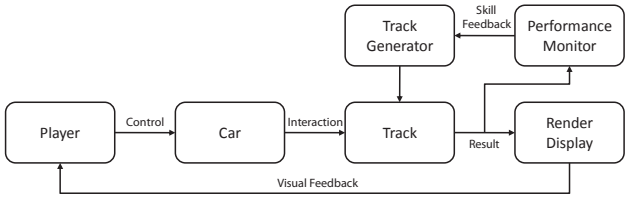

Figure 1: Overview of the proposed system.

This section presents the core components of the proposed game system (see Fig. 1). The system consists of two real-time feedback loops: the main loop is a classic player-game loop with the output (i.e., feedback to the user) being displayed on the screen. There is also an additional loop consisting of a module which assesses the player's performance and feeds this information back to the track generator, resulting in an adaptively adjusted track which in turn affects the player actions.

The Car Model

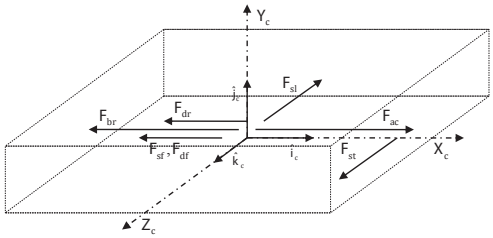

Figure 2: The model of the car used in the simulation together with all forces implemented.

The car is simulated as a rigid cuboid of length l_c, width w_c, height h_c and mass m_c approximately matching that of a real car (see Fig. 2). The forward and backward movement of the car is controlled by applying an acceleration force $\boldsymbol{F}_{ac}$ along the local X_c axis, which is generated by a simulated automatic gearbox. To simulate brakes, a braking force $\boldsymbol{F}_{br}$ is applied in opposite direction to the car movement. To improve the responsiveness of brakes for higher velocities, a modified formula for $\boldsymbol{F}_{br}$ that takes into account the car speed v_c has been used. The car is controlled by a steering force $\boldsymbol{F}_{st}$ proportional to the car speed v_c and applied to the front of the car along the Z_c axis. The other simulated forces consist of gravity $\boldsymbol{F}_g$ and resistive forces including air drag $\boldsymbol{F}_{dr}$, static friction $\boldsymbol{F}_{sf}$, dynamic friction $\boldsymbol{F}_{df}$ and sliding friction $\boldsymbol{F}_{sl}$ to simulate skidding. All forces with an exception of $\boldsymbol{F}_{st}$ are applied to the car's centre of mass. Table 1 presents detailed formulas for each modelled force while Table 2 presents parameter values used in the simulation.

Table 1: Simulated forces, where v_c denotes the car speed and $\hat{\boldsymbol{i}}_c$, $\hat{\boldsymbol{j}}_c$, $\hat{\boldsymbol{k}}_c$ define the local car coordinates.

Force	Descritpion
$\boldsymbol{F}_g = -gm_c\hat{\boldsymbol{j}}$	gravity (global force)
$\boldsymbol{F}_{ac} = F_{ac}\hat{\boldsymbol{i}}_c$	acceleration force
$\boldsymbol{F}_{br} = -c_{br}v_c\hat{\boldsymbol{i}}_c$	braking force
$\boldsymbol{F}_{st} = \pm c_{st}v_c\hat{\boldsymbol{k}}_c$	steering force
$\boldsymbol{F}_{dr} = -c_d v_c^2\hat{\boldsymbol{i}}_c$	air drag
$\boldsymbol{F}_{sf} \leq -\mu_s gm_c\hat{\boldsymbol{i}}_c$	static friction
$\boldsymbol{F}_{df} = -\mu_d gm_c\hat{\boldsymbol{i}}_c$	dynamic friction
$\boldsymbol{F}_{sl} = F_{sl}\hat{\boldsymbol{k}}_c$	sliding friction

Table 2: Physical parameter values used in the simulation.

Parameter	Name	Value
g	gravitational constant	9.8 m/s^2
l_c	car length	4.45 m
w_c	car width	1.70 m
h_c	car height	0.95 m
m_c	car mass	1182 kg
c_{br}	braking coeff.	10^5 kg/s
c_{st}	steering coeff.	$3 \cdot 10^4$ kg/s
c_d	air drag coeff.	300 kg/m
$\mu_s = \mu_d$	friction coeff.	0.3
F_{sl}	sliding force	10^6 N

The Track Model

The track is composed of alternating segments, called "straights" and "curves" (see Fig. 3). Straight segments are simply rectangles with varying length l, width w and gradient α parameters. Curved segments are partial annuli with varying turn radius l, turn angle θ, width w, gradient α and camber β parameters. The gradient of a segment α is the slope of the road measured along its length; on a segment with a positive gradient, the start of the segment is lower than the end. It is measured as the angle the length makes with the horizontal plane. The camber β of a curve is the slope of the road measured across its width; on a curve with a positive camber, the inside of the curve is lower than the outside. It is measured as the angle the width makes with the horizontal plane. All segment parameters are embedded into a single vector $\boldsymbol{x}_s = \{l, \theta, w, \alpha, \beta\}$ which is generated in real-time by the track generation algorithm as the game progresses.

Track Generation

The track generation algorithm continuously generates alternating straight and curved segments according to Alg. 1. After each segment, the track parameters are ad-

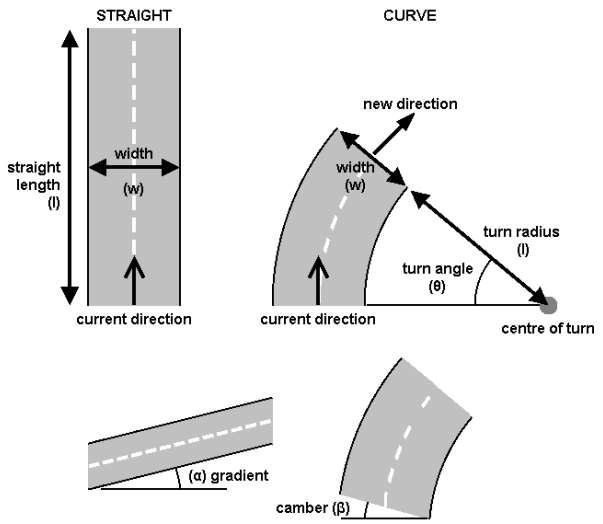

Figure 3: Track segments and their parameters.

Algorithm 1: Track generation algorithm.

1. Initialise all segment parameters $\boldsymbol{x}_s$ with default values.
2. Generate a new straight segment $\boldsymbol{x}_s = \{l, -, w, \alpha, -\}$.
3. Adjust the track parameters according to Algorithm 2.
4. Generate a new curve segment: $\boldsymbol{x}_s = \{l, \theta, w, \alpha, \beta\}$.
5. Adjust the track parameters according to Algorithm 3.
6. Go to Step 2.

justed according to the estimated player's performance which takes into account a skill level metric and selected events that might have occurred over the course of the completed segment. Values for l and θ are generated from probability distributions - uniform in the presented case, but could be of any type in principle. The distribution parameters for turn angle can be determined from other variables: $\theta_{min} = l_{MIN}/l$ and $\theta_{max} = \theta_{MAX}$, while l_{min} and l_{max} are being incrementally adjusted together with the values of segment parameters w, α and β according to Algorithms 2 and 3.

The presented algorithm modifies the track variables based on a set of heuristics which follow observations that the main difficulty in racing games arise from bendy, narrow tracks of variable turn directions. Gradient α depends on calculated skill level p (see Skill Level Estimation Section for details), while other parameters are adjusted when the player hits one of the

Algorithm 2: Track adjustment after the straight segment.

- Width adjustment: $w = w \pm \Delta w$. If the player hit the edge of the track, increase w, otherwise decrease w.
- Gradient adjustment: $\alpha = \alpha \pm \Delta\alpha$. Calculate skill level p (see Skill Level Estimation Section for details). If p exceeds the specified threshold p_t decrease α, otherwise increase α.
- Generate new turn radius and angle from the uniform distribution: $l = \mathcal{U}(l_{min}, l_{max})$, $\theta = \mathcal{U}(\theta_{min}, \theta_{max})$.

walls, or/and drives with the speed different to the nominal speed v_n. The nominal speed $v_n = [\sqrt{l \cdot n_{lo}}, \sqrt{l \cdot n_{hi}}]$ is an experimentally derived measure that defines an "appropriate" speed interval for a given length/radius of a curve segment.

Algorithm 3: Track adjustment after the curved segment.

- Minimum length/radius adjustment: $l_{min} = l_{min} \pm \Delta l_{min}$. If the player hit the outside wall with the speed higher than the nominal speed v_n, increase l_{min}, otherwise decrease l_{min}.
- Maximum length/radius adjustment: $l_{max} = l_{max} \pm \Delta l_{max}$. If the player approached the corner with the speed lower than the nominal speed v_n decrease r_{max}, otherwise increase r_{max}.
- Camber adjustment: $\beta = \beta \pm \Delta\beta$. If the player hit the inside wall, decrease camber (i.e., make road lean more towards the outside of corners). If the player hit the outside wall with the nominal speed v_n, increase camber (i.e., make road lean more towards the inside of corners). If the player did not hit a wall, reduce absolute camber.
- Gradient adjustment: $\alpha = \alpha \pm \Delta\alpha$. Calculate skill level p (see Skill Level Estimation Section for details). If p exceeds the specified threshold p_t decrease α, otherwise increase α.
- Generate new length value from the uniform distribution: $l = \mathcal{U}(l_{min}, l_{max})$.

Parameter adjustment can be performed in a number of ways. Here, three methods are proposed:

1. Linear Adjustment: a constant value (different for each variable) is added or subtracted to that variable, e.g., $\Delta\alpha = c_\alpha$.

2. Proportional Adjustment: a constant value (different for each variable) is multiplied or divided by that variable, e.g., $\Delta\alpha = \alpha(c_\alpha - 1)$. This method is not applicable to camber and gradient where both positive and negative values are used and therefore these parameters will be adjusted using Linear Adjustment.

3. Width-weighted Linear Adjustment: a constant value (different for each variable) is multiplied by the normalised width value before being added or subtracted to that variable, e.g., $\Delta\alpha = c_\alpha(w - w_{MIN})/(w_{MAX} - w_{MIN})$. This method should place a greater importance on avoiding the walls.

Skill Level Estimation

The skill level p is calculated by comparing the minimum and maximum values of length/radius, width and camber against their respective absolute values. A highly skilled player will have l_{min} and w close to their lower limits, l_{max} at its upper limit, and $\beta = 0$. Conversely, a less skilled player will have l_{min} and w close to their upper limits, l_{max} at its lower limit, and $\beta = \beta_{MAX}$. The resulting values are then normalised and their weighted sum is used as the skill level for a given segment.

Implementation Details

The described components were integrated into a car racing game implemented using the NVIDIA PhysX library for physics simulation (ver. 2.8.4) and OpenGL for basic rendering. Each track segment has a wall on both left and right sides, perpendicular to the road of the track itself in order to prevent the player from leaving the track. The graphical model of the car is based on Lotus Evora. Since the generated track could be of arbitrary length, only part of the track is kept in memory: this includes two generated segments ahead and two previous ones with respect to the current segment. To avoid misalignments between the segments when $\beta \neq 0$, there is also an additional short linking component that connects the vertices of adjoining segments.

EXPERIMENTS

Data Collection

A random sample of 16 students and lecturers at the University of Lincoln were asked to play four tracks, each consisting of 50 segments and taking approximately 2:30 min. to complete. Three of the tracks are generated adaptively, using each of the three adjustment methods listed previously, and the fourth pre-generated control track that does not adapt to the player's actions. The tracks are presented in a random order to the participants so as to prevent bias.

After completing all four tracks, each of the participants was asked to complete a questionnaire asking participants to rate each track's "fun" and "challenge" level and also to select their preferred track. The questionnaire also asked participants about the frequency with which they play racing games (i.e. number of titles played in the last six months) as well as how proficient they felt they are at racing games. The ratings were expressed in five-point Likert scale.

Results

Table 3: User responses to the questionnaire with respect to tracks generated using different adjustment methods (1-3) and the control track (C).

	Adjustment Method			
Rating	1	2	3	C
Fun	3.88	3.56	2.75	2.50
Challenge	3.44	3.36	3.56	2.44
Preference	8	4	3	1

The summary of results from the questionnaire are presented in Table 3. On average, the tracks generated using Linear Adjustment have the highest average "fun" value, and the control track the lowest. The Proportional Adjustment method had a similar value to that of Linear Adjustment, whereas the Width-weighted Adjustment tracks scored comparably to the control track. This suggests that the width-weighted method contributed less to the perceived fun factor than the other two methods. The average difficulty remains about the same throughout the tracks generated adaptively. This was expected, as the track generation attempts to match the player's skill level. The control track is lower in difficulty, as it does not adapt to the player's skill level. This may suggests that either the control track is too easy or the adaptive tracks are too difficult for the average player. Majority of the participants preferred the tracks generated using Linear Adjustment, which indicates that this is the most suitable track generation method to satisfy the player's gaming experience.

Figure 4 presents the mean skill level p for a given questionnaire response type and its value. Four different skill levels were calculated; the "challenge" parameter applies to the track the skill level was calculated for, whereas the "frequency" and "proficiency" responses are simply duplicated across the four tracks.

On tracks which participants felt that were more challenging, their calculated skill level was lower, and vice versa. However, given that the track should adapt to the player's skill level, the challenge should remain approximately constant, or at least have little correlation. This suggests that while the track is adapting, it is not

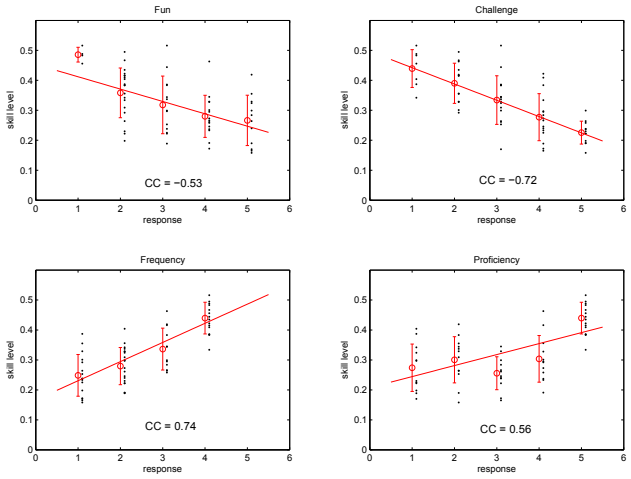

Figure 4: Skill level with respect to different response type and value (CC - sample correlation coefficient).

doing so enough with respect to both the beginners and expert players.

As there was a strong correlation between skill level and playing frequency, it can be said both that frequency of playing racing games is a good indication of skill level, and that the skill level calculation is an accurate assessment of the player's ability. This shines a somewhat interesting light on how players perceive their skill level, as there is a much lower correlation between skill level and proficiency. Many racing games use a difficulty level to allow the player to choose for themselves; it may be that this is a flawed idea, as players may not be able to accurately determine their own skill level.

It is worth noting that no conclusion can be drawn from these results on the comparative playing experience between an adaptive track and a well-designed prescribed track. However, a designer can only make a finite number of tracks, thus making adaptive track generation a more viable choice for creating a very large number of tracks, or for creating a track which has no ending.

CONCLUSIONS

The conducted experiments demonstrate that the proposed method, despite its relative simplicity, is a feasible option for generating adaptive racing tracks in real-time. It seems that adaptive track generation in racing games can provide the player with a greater level of "fun factor" than a non-adaptive track generation, but more research is needed in order to confirm its effectiveness in replacing/assisting a human designer. Other shortcomings of the presented work include a relatively low sample of the collected data which may have significantly affected some of the derived conclusions; data collected from an on-line release of the game would allow for more reliable results. The game itself could be integrated into the existing racing simulators (e.g., (TORCS 2005)) that could lead to more experimental data available, but also would enable enhanced physics modelling and additional game functionality. The presented adaptive models are ad-hoc and manually crafted for a generic user. The model parameters (weights, increments, etc.) could be learnt directly from data and tailored to a specific user type, brining it closer to other work on adaptive racing tracks (e.g. (Togelius et al. 2006)).

REFERENCES

R. Bernhaupt. *Evaluating User Experience in Games: Concepts and Methods*. Springer Publishing Company, Incorporated, 1 edition, 2010.

M. Booth. The AI Systems of Left 4 Dead. In *Keynote, Artificial Intelligence and Interactive Digital Entertainment Conference*, Palo Alto, CA, USA, 2009.

E. J. Hastings, R. K. Guha, and K. O. Stanley. Automatic Content Generation in the Galactic Arms Race Video Game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.

Microsoft Research. Drivatar Technology. Website, 2012. URL `http://research.microsoft.com/en-us/projects/drivatar/default.aspx`.

C. Pedersen, J. Togelius, and G. N. Yannakakis. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

J. Togelius, R. D. Nardi, and S. M. Lucas. Making racing fun through player modeling and track evolution. In *Proc. of the Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.

J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Trans. Comput. Intellig. and AI in Games*, 3(3):172–186, 2011.

TORCS. The Open Racing Car Simulator. Website, 2005. URL `http://www.torcs.org`.

J. Whitehead. Toward proccedural decorative ornamentation in games. In *Proc. of Workshop on Procedural Content Generation in Games*, PCGames '10, pages 9:1–9:4, New York, NY, USA, 2010. ACM.

G. N. Yannakakis and J. Hallam. Real-time Game Adaptation for Optimizing Player Satisfaction. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(2):121–133, June 2009.

G. N. Yannakakis and J. Togelius. Experience-driven procedural content generation. *T. Affective Computing*, 2(3):147–161, 2011.